スーパー ビギナーズ
超初心者 Super Beginners' 講座 第4回

前回予告したとおり、今回の講座はスプライトパターンの作り方を教えるぞ。パターンの作り方さえ覚えてしまえば、MSXがいっそう楽しくなるはずだ。

スプライトパターンの作り方

ファンダムに掲載されているゲームなんかのプログラムを打ちこんでいるとき、よく目にするのが「SPRITE$」とか「PUTSPRITE」という命令だろう。

この命令についてマニュアルを読んでみると、「スプライト」という言葉が出てくる。

スプライトとは、画面に表示されている文字なんかに影響されずに、画面のなかを飛びまわることのできる特殊なものだ。

こいつはだれにでもかんたんに使うことができて、しかもこいつを使うと、けっこうかんたんにゲームが作れたりする。

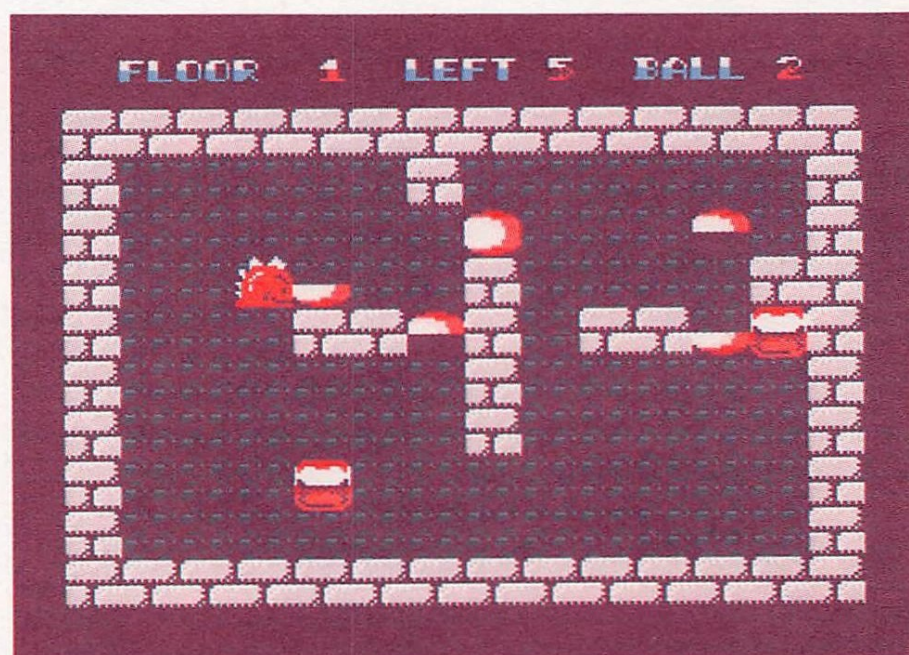
シューティングゲームを作りたいなあ、とか思ったら、スプライトの出番だ。

せまりくる敵機やカッコイイ自機、ミサイルなんかもみんなスプライトで作れるのだ。

RPGでもパズルでも、動くものならスプライトを使うととても便利なんだ。

■パターンデータを作る

1枚のスプライトは8×8の点で構成されている。

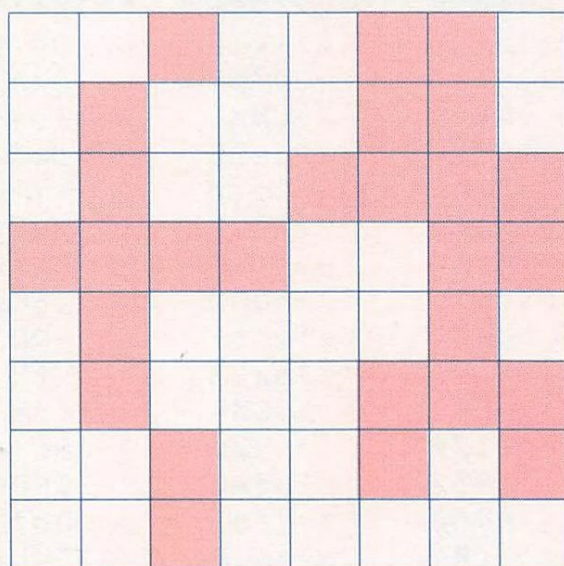
だから、スプライトにしたいもの、例えば飛行機なんかを8×8のマス目に書いてみよう。

↑64ページに掲載されている「TAR-BOT2」のゲーム画面。主人公のキャラクタは、スプライトで表示されているから、自由に動きまわることができるのだ

じつはこれさえできればあとはかんたんで、点にしたところに「1」、点のないところに「0」と置き換える。

つぎに横1列ずつ取り出してあたまに「&B」をくっつける。

これでスプライトパターンデータのできあがり。

■パターンの作り方

パターンの元図を作る

方眼紙などの紙か、ふつうの紙に8×8のマス目を書いたものを用意し、その中を塗りつぶしてキャラクタをデザインしていく。8×8のマス目を有効に使おう。

元図に1とØの数字をふる

塗りつぶしたマスには「1」を書きこみ、空白のままのマスには「0」を書きこんでいく。別の紙におなじ並びで数字だけを書いておいてもいい。

データに変える

&BØØ1ØØ11Ø
&BØ1ØØØ11Ø
&BØ1ØØ1111
&B1111ØØ11
&BØ1ØØØØ1Ø
&BØ1ØØØ111
&BØØ1ØØ1Ø1
&BØØ1ØØØØØ

横1列ごとに数字を並べていき、そのあたまに「&B」という文字を付ければ、データのできあがり。

スプライトパターンを定義する

作ったスプライトパターンデータを、MSXに登録しよう。

どうやって登録するのか？

ここで、「SPRITE$」というものが登場するのだ。

まず、下の写真のように「PRINT」と打ちこんだあと、続けてスプライトパターンデータを打ちこみ、実行しよう。

すぐ下に数字が表示される。

こうして8つとも数字にして、順番にメモしておこう。

全部数字にできたら、

SPRITE$(Ø)=CHR$(1列目のデータ)+CHR$(2列目のデータ)+……CHR$(8列目のデータ)

というふうに、「SPRITE$(Ø)=」のあとに、メモした数字を「CHR$(」と「)」の間に入れ、上の列のデータから、順番に足し算の形にして打ちこんでいくのだ。

全部打ちこんだらリターンキーを押して登録完了。

ところでファンダムに掲載されたプログラムでは、下のリストのようなスプライトパターン定義をしていることがある。

PRINT CHR$(38)

を実行すると「&」と表示されるように、「&」も「CHR$(38)」も同じ文字なので、プログラムを短くするために、あらかじめ文字に変えたものを登録しているのだ。

!! キーボードにふれてみよう①

```
SPRITE$(Ø)="&FO&BG% "
```

```
PRINT &B00100110
 38
Ok
█
```

スプライトを表示してみよう

さあ、それではスプライトを画面に表示してみよう。

スプライトを表示させるには、「PUTSPRITE」という命令を使うのだ。

ただし、下の写真やリストにあるように、「PUTSPRITE」のあとに、いくつか数字を付ける必要がある。

どこにスプライトを表示するかを指示するのが、かっこのなかの座標だ。

かっこの次の数字でスプライトの色、その次の数字がスプライトの形を指定している。

SCREEN1であることを確認してから、「キーボードにふれてみよう」の①と②を実行してみよう。左ページにあるスプライトが画面に表示されるぞ。

!! キーボードにふれてみよう②

```
PUTSPRITE Ø,(1ØØ,1ØØ),8,Ø
```

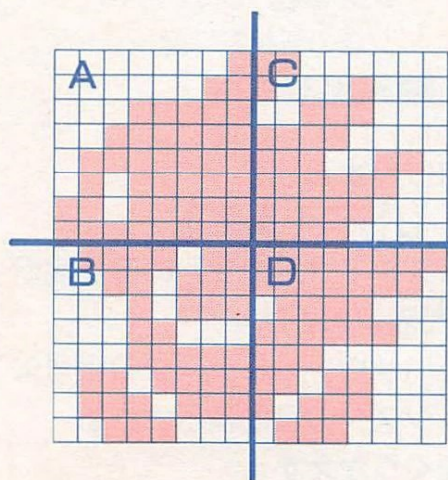
⬆左のページの下で作った8×8ドットのスプライトを、SPRITE$とCHR$を使って定義し、PUTSPRITE命令で表示しているところ。赤く表示されたのがスプライトだ

16×16ドットのスプライトの作り方

スプライトは8×8の点でできていると教えたが、じつはSCREEN命令によって大きさを変えることができるのだ。

SCREEN1, 2

と実行すると、16×16の点でできたスプライトも作れるのだ。

■16×16ドットのスプライトパターンデータの作り方

16×16のスプライトを作る方法は、8×8のスプライトとほとんど変わらない。

元になる絵を16×16のマス目に書き、点のあるなしによって1と0を振っていく。

次がちょっと違うのだが、左下の図のように、8×8の大きさに4分割してしまう。

そして、それぞれ8×8のスプライトを作る要領で、数字にしていくのだ。

さて、問題はスプライトパターン定義の部分にある。

スプライトパターン定義を、数字にしたデータをCHR$関数の中に入れて、SPRITE$のあとに並べておこなうのには、変わりはないのだが、4つに分けたスプライトパターンデータを、左上、左下、右上、右下の順に並べ、32個の数字をすべて並べて定義しなければいけない。

詳しいやり方は左下に書いてあるので、よく読んでおいてほしい。

A C
B D

■データはABCDの順にならべて定義する

左図でいうとデータをABCDの順に並べて定義するのだが、ほとんどの場合、SPRITE$(Ø)=CHR$(1)+……と続けて打ちこんでいっても長すぎてうまくいかない。右の写真のようにプログラムを組んで定義するのが確実だ。文字変数に1文字ずつ足していき、その文字変数で定義するのだ。DATA文にパターンデータを順番どおりに並べるだけだから、かなり楽になる。

⬆16×16ドットのスプライトはプログラムを組まないとうまく定義できない。写真のプログラムを打ちこんで、行20のデータのところに、32個の数字を打ちこめば定義できる

ファンダムのゲームを遊んだあと、画面のなかに不思議な文字を見かけることがある。今回の講座では、この不思議な文字の正体を解き明かそう。

不思議な文字の正体

ファンダム掲載プログラムを打ちこみ、RUNしてじゅうぶん満足したあと、プログラムを止めてみると、画面にはグラフィックのような不思議な文字が残っていることがある。

はじめのうちは気にならなくても、少しずつMSXに慣れてくると、「これはいったいなんだろう」と悩んだりする。

また、この文字のせいで、エラーが出ても何がなんだかわからないときがあったりする。

今月のファンダムに掲載された「SUBMERGE」を例に、不思議な文字の正体を明かそう。

■不思議な文字の正体は？

SCREEN1を実行したあと、右上にある「キーボードにふれてみよう①」を実行しよう。

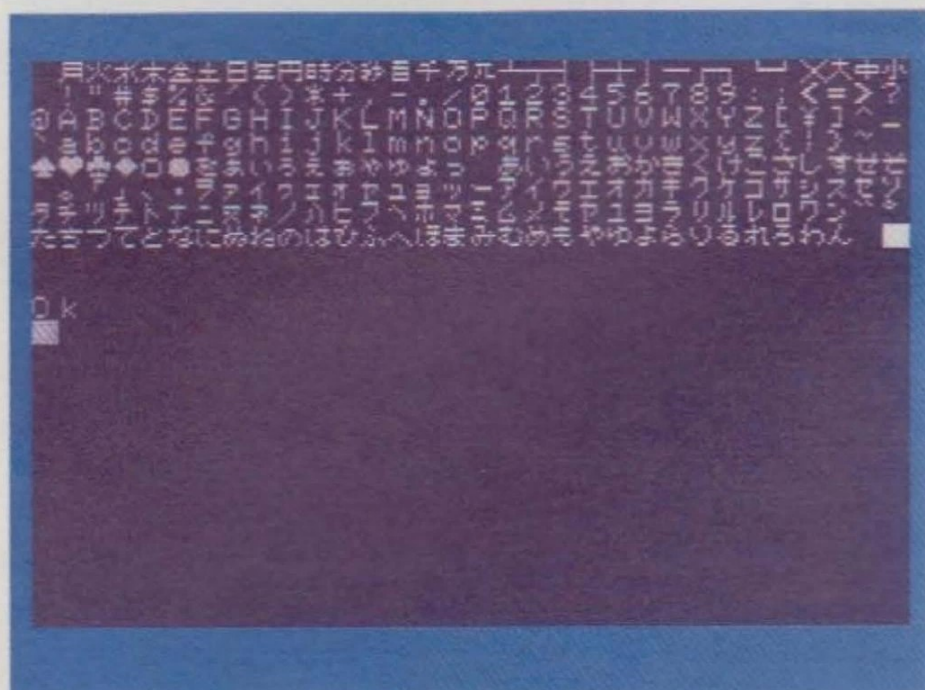
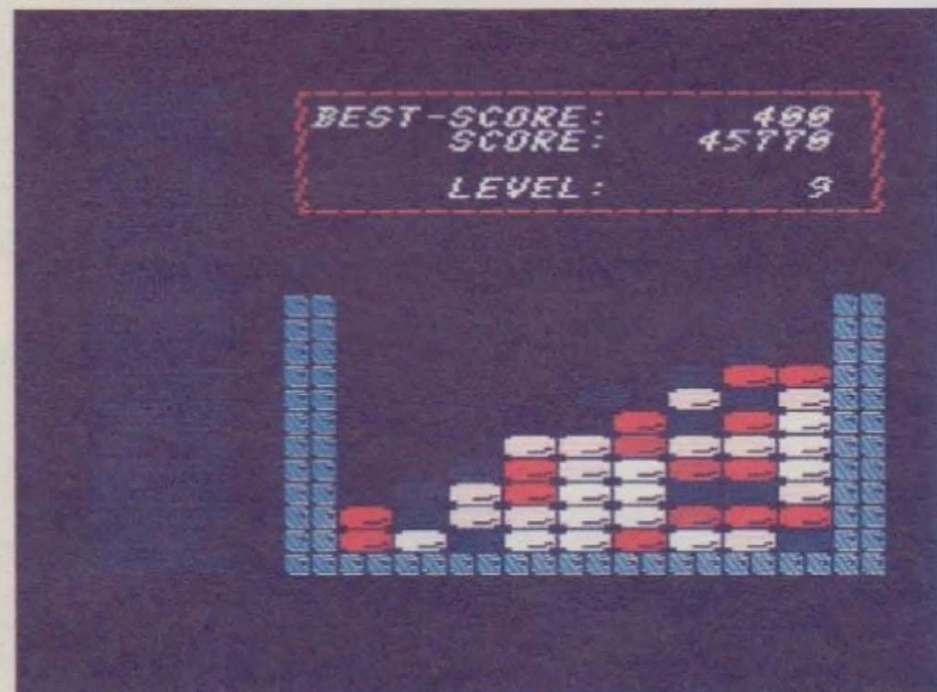
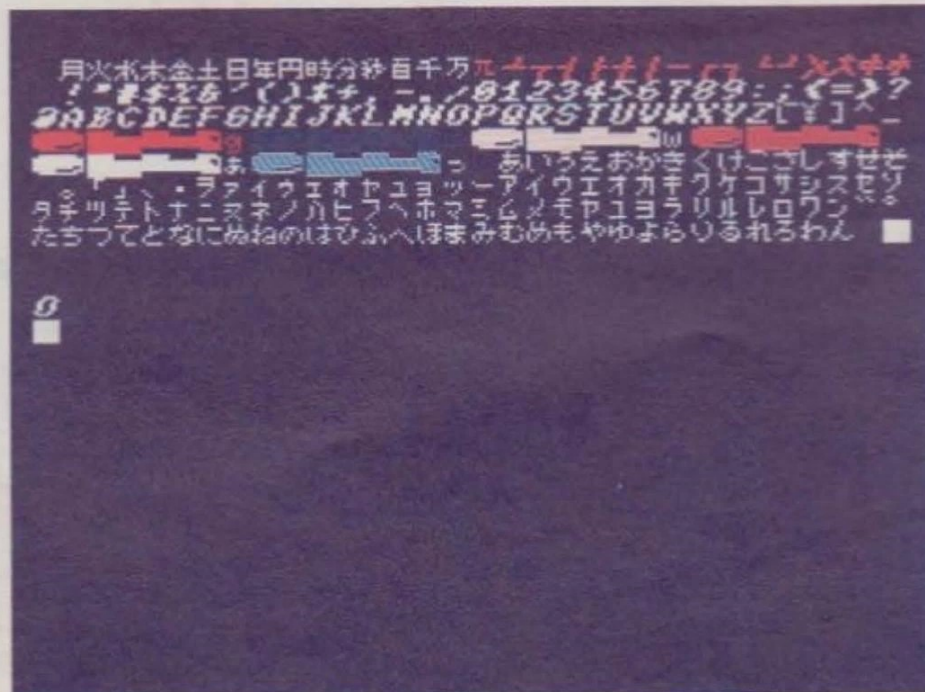
すると画面には、MSXで打ちこむことができる文字がすべて表示される(右上の写真)。

つぎに「SUBMERGE」を打ちこんで走らせたあと(下の写真)、CTRLキーとSTOPキーをいっしょに押してプログラムを止め、「キーボードにふれてみよう①」を実行してみよう(右下の写真)。

そこにはさきほどとは異なった、不思議な文字がたくさん表示されているはずだ。

ためしに、キーボードから「A」と「a」の2つの文字を打ちこんでみるとわかるが、文字の形が変わって不思議な文字になっているのだ。

‼キーボードにふれてみよう①

```
CLS:FORI=0TO255:VPOKE&H1800+I
,I:NEXT:LOCATE0,10
```

SCREEN1を実行したあと、上の「キーボードにふれてみよう①」を実行したところ。グラフィックキャラクタを含む、すべての文字がきちんとならんで表示される

今月のファンダム(64ページ)に掲載されている「SUBMERGE」のゲーム画面。横長の丸いブロックや、よく見るとアルファベットが斜体になっているぞ

「SUBMERGE」をゲームのとちゅうで止めて、「キーボードにふれてみよう①」を実行したところ。上下2枚の写真をよく見くらべてみよう

VRAMだけが知っている

不思議な文字の正体は、なんてことはない、文字の形が変わっていただけのことだったのだ。

しかし、どうやって文字の形を変えているのだろうか。

■文字の形の情報は、VRAMに格納されている

MSXの文字の形を変えるには、SCREEN1の画面がどういう構造になっているのか、少し知っておく必要がある。

MSXにはVRAMというものがあり、そこには画面に関する情報が保存されていて、文字やスプライトの形や色、どこに何が表示されているかなどの情報が保存されているのだ。

つまり、文字の形を変えるには、VRAMに保存されている文字の形の情報を変えてあげればいいことになる。

しかし、VRAMに格納されている文字の形の情報を変えるといっても、どこをどう変えればいいのかがわからなくては、とてもできそうにない。

そこで、下にSCREEN1のときのVRAMは、いったいどうなっているのかをわかりやすく解説しているので、読んでおいてほしい。VRAMは知れば知るほどおもしろいのだ。

■キャラクタパターンジェネレータテーブルを捜せ

文字の形の情報は、VRAMのキャラクタパターンジェネレータテーブルという、やたら長い名前の場所に保存されている。

名前がわかっても、場所がわからなきゃしょうがない。

保存場所は、文字ごとに決まっているキャラクタコードを8倍すればいいのだ。

なにもここで悩まなくても、MSXが教えてくれるのだ。

たとえば「¥」のキャラクタパターンジェネレータテーブルのアドレスなら、

PRINT ASC("¥")*8

を実行すれば、一発でわかってしまうのだ。便利、便利。

■キャラクタパターンデータを作ろう

書き換えるアドレスがわかったら、今度はデータの番だ。

MSXの文字は、8×8の点で構成されているので、あらかじめどんな形の文字を作るのかを、8×8のマス目のある方眼紙などに、デザインしてみよう。

下の写真のように、点のあるところは「1」に、点のないところは「0」にしてデザインしていくと手間がはぶけるが、はじめのうちは、点のあるところだけ軽く塗りつぶしていき、デザインが終わってから「1」と「0」を振り分けていくといい。

これでキャラクタパターンデータは完成したといえる。

SCREEN1のVRAM構成

アドレス	内容
&H0000〜&H07FF	キャラクタパターンジェネレータテーブル(2048バイト)
&H1800〜&H1AFF	パターン名称テーブル(768バイト)
&H2000〜	カラーテーブル(32バイト)
&H3800〜&H3FFF	スプライトパターンジェネレータテーブル(2048バイト)

SCREEN1のVRAMの構成図。ほとんどの部分は使われていないあきエリアだ。

SCREEN1のVRAM構成は、ほぼ左の図のようになっている。文字の形、色、表示位置の情報と、スプライトの形、色と表示位置の情報、パレットの情報などがあるのだが、ここでは以下のおもなものについて説明しよう。

■キャラクタパターンジェネレータテーブル

MSXで使える文字の、形の情報が格納されているところ。

アドレスは0からはじまり、キャラクタコード順に1文字につき8バイトずつとられている。

ある文字のキャラクタパターンデータが格納されているアドレスは、その文字のキャラクタコード×8の場所から8バイトとなる。

右にある「キーボードにふれてみよう②」を実行すると、「A」のキャラクタパターンデータが表示される(右の写真)。

リストの「"A"」の部分のAを他の文字に変えて試してみよう。

■パターン名称テーブル

画面に表示されている文字が格納されているところ。

アドレスは&H1800から768バイトの範囲。

右の「キーボードにふれてみよう③」を実行すると、画面に表示されている文字のデータがわかるので、やってみよう。

■カラーテーブル

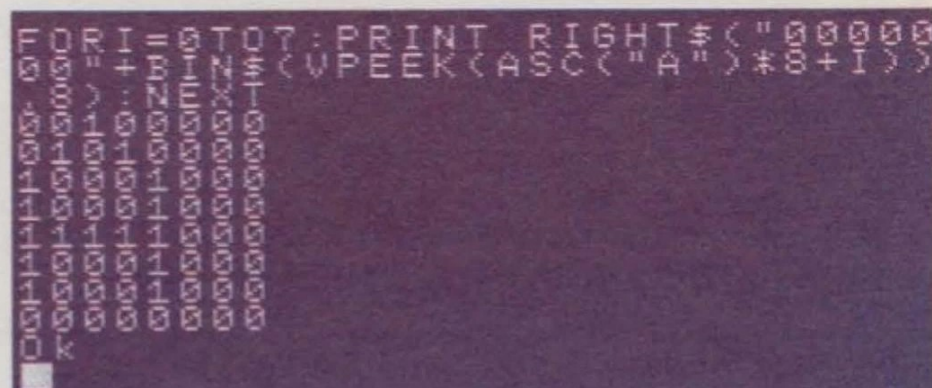
文字を表示するときの、色の情報を格納しているところ。

アドレスは&H2000からの32バイトとなっていて、キャラクタコード順に8文字につき1バイトとなっている。

「キーボードにふれてみよう①」を実行したあと、右にある④を実行すると、「A」を含む8文字の色が変わるのがわかる。

はじめの部分の"A"やつぎにある数字を変えて試してみよう。

キーボードにふれてみよう②

```
FORI=0TO7:PRINT RIGHT$("00000
00"+BIN$(VPEEK(ASC("A")*8+I))
,8):NEXT
```

「キーボードにふれてみよう②」の実行画面。アルファベットの「A」の形がよくわかる。このようにMSXはVRAMに文字の形を記憶しているのだ

キーボードにふれてみよう③

```
10 SCREEN1:WIDTH32:KEYON:SPRI
TE$(0)=CHR$(255)+"♥♥♥♥♥♥"+CHR
$(255):FORI=0TO1:PRINTINKEY$;
:X=POS(0):Y=CSRLIN:PUTSPRITE0
,(X*8,Y*8-1),15,0:KEY1,STR$(V
PEEK(&H1800+Y*32+X)):I=-STRIG
(0):NEXT
```

キーボードにふれてみよう④

```
A$="A":F=8:B=1:VPOKE &H2000+A
SC(A$)¥8,F*16+B
```

いろんな文字をつくろう

VRAMとはいったいどういうものなのか、あるていどわかったら、いよいよ自分だけのオリジナル不思議文字を作ろう。

■キャラクタパターン定義

いよいよできあがったキャラクタパターンデータを、VPOKEという命令を使って、書き換えるべきアドレスに登録(キャラクタパターン定義)しよう。

ここでちょっと右にある「キーボードにふれてみよう⑤」を打ちこんでみよう。

行20~90のDATA文にはさきほど作ったキャラクタパターンデータと似たものが並んでいる。

このリストを実行すると、「a」という文字が、レンガの模様に変わるのだ。

それでは、さきほど作ったキャラクタパターンデータを、行20からのDATA文のところに打ちこんでいき、行10の「a」の部分を書き換えたい文字に変えて実行してみよう。

うまくいったかな?

キャラクタパターン定義は、書き換えるアドレスのところから、順番にキャラクタパターンデータを、VPOKEを使って書きこんでいくだけなのだ。

ただし、さきほど作ったキャラクタパターンデータは文字データなので、あたまに「&B」を付けて数値に変える必要がある。

また、このリストを実行すると、書き換わった文字の色も赤くなっただろう。

SCREEN1では、このように8文字ごとに色を変えることもできるのだ。

この色の設定のしかたについては今回はふれないが、興味のある人は、行10の最後のほうにある「&H68」の部分を、0から255の範囲の数値に変えて実行してみるといい。

そうそう、SCREEN命令を実行すると、文字の形などがもとの状態にもどるぞ。

❶「キーボードにふれてみよう⑤」の実行後、「a」を画面に適当に書いたもの。これが「a」とはとてもわからない

‼キーボードにふれてみよう⑤

```
10 A=ASC("a"):FORI=0TO7:READA
$:VPOKEA*8+I,VAL("&B"+A$):NEX
T:VPOKE&H2000+A¥8,&H68
20 DATA 00100000
30 DATA 00100000
40 DATA 00100000
50 DATA 11111111
60 DATA 00000010
70 DATA 00000010
80 DATA 00000010
90 DATA 11111111
```

スプライトとキャラクタの関係

前回の講座を読んだ人はピンときたかもしれない。

つまり、スプライトも不思議文字も、パターンデータの作り方はおなじなのだ。

ただ、ちょっと違うのが、パターンの定義のしかたなんだが、これは、スプライトはSPRITE$という命令があったが、不思議文字にはとくに何もない。

じつは、スプライトパターン定義も、キャラクタパターン定義のように、VPOKEを使って直接定義してしまう方法もあるのだ。

また、ファンダムのゲームではスプライトとキャラクタが仲良く手を結んでゲームにしているが、スプライトが文字と重なっても文字が隠れるだけで消えないし、CLSをして画面消去してもスプライトは消えない。

掲載プログラムでは、この水と油のような性格の文字とスプライトを、どう料理しているのだろう。

左の写真は「SKELETON」のゲーム画面だが、ここでも文字とスプライトは仲良くしている。

ではどういうふうにして、仲良くさせているのか、ちょっとプログラムをのぞいてみよう。

上の写真の部分がそれで、じつはスプライトの表示されている座標をもとに、そこに何があるのかVRAMのパターン名称テーブルを調べているのだ。

文字を表示する位置はテキスト座標といい、スプライトを表示する位置はスプライト座標という。

スプライト座標(X, Y)にあるスプライトは、テキスト座標(X/8, (Y+1)/8)の位置の文字と重なっていることになる。

反対にテキスト座標を(X, Y)として考えると下図のようになる。

次回の講座では、不思議文字とスプライトとの関係も含めて、画面表示に関することを、ちょこっと講義してみたいと思う。

```
list 5
5 F=-(F=0):V=VPEEK(6720+(X+4)¥8):IFY=0TH
ENIFV>32THENX=X+(V-61)*2:H=0ELSEY=-1
Ok
?&H1800+18*32
 6720
Ok
```

❶「SKELETON」のプログラムリストの行5を調べてみると、スプライト座標に使っているはずの変数Xを使って、VRAMのパターン名称テーブルを調べているぞ

❶これも今月のファンダム(59ページ)に掲載されている「SKELETON」のゲーム画面。スケルトンと黒い鉄球はスプライトだけど、地面は文字。どう判定しているのか?

■スプライト座標とテキスト座標の関係

Ⓐの位置の文字と、Ⓑの位置にあるスプライトとはぴったり重なっている。

```
Ⓐ LOCATE X,Y
Ⓑ PUTSPRITE 0,(X*8,Y*8-1),15,0
```

超初心者 Super Beginners' 講座 第6回

スーパー ビギナーズ

画面のなかで、ふにょふにょ動くもの。それはきっとスプライトだ。このスプライトというヤツは、奇妙にも文字をよけたりなんかする。今回はその仕組みに触れる。

スプライトを定義する

今回の講座では、プログラム作りにも少し挑戦してみよう。

具体的には、「キーボードにふれてみよう」の①と③〜⑤を続けて打ちこむと、ひとつのプログラムになるというものだ。

それぞれのちょっとしたリストはひとつの目的があってできているので、プログラムを作るときの参考にしてほしい。

■まず、スプライトパターンを定義して用意する

今回の講座は、スプライトの表示について触れてみよう。

スプライトの表示とひとことにいっても、そこにはさまざまな処理がからまってくるものだ。

まあ、とりあえず、主役となるスプライトを用意しておく必要があるので、「キーボードにふれてみよう①」を打ちこんでほしい。

このリストには、いくつかの命令が入っているが、ここで重要なのは「SPRITE$」という命令だ。

これを「RUN」と打ちこんで走らせると、画面はSCREEN1になり、スプライトパターンが定義される。

このスプライトは今月採用された「オイルショック'91」(右下の写真)から、ちょっと拝借してきたものだ。

つぎに、「キーボードにふれてみよう②」を打ちこんで実行して、右の写真のように水鳥が表示されるか確認しよう。

うまく表示されたかな？

ここでつまずいていたらつまらないので、ちゃんと表示されるようになるまで、①と②を打ちこみなおそう。

ところでこの水鳥、これから最後まで付き合ってくれる主役となるものだから、わたしの独断で「カルちゃん」と名前を付けておこう。

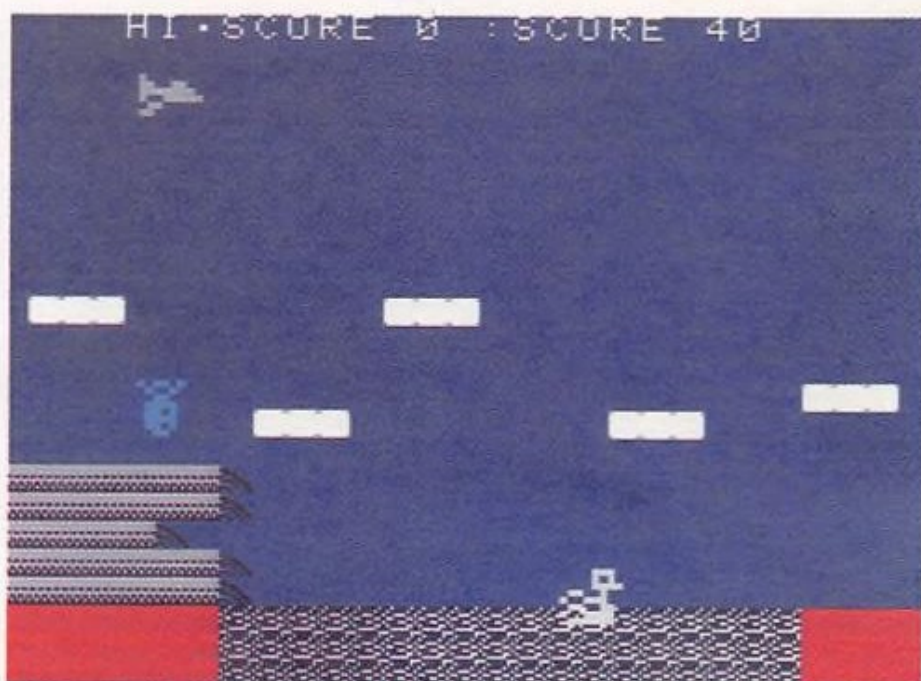
さあ次は、カルちゃんの遊ぶ舞台を作るぞ。

!!キーボードにふれてみよう①

```
10 SCREEN1,1:WIDTH32:COLOR15,4,7:CLS:FOR
J=0TO1:B$="":READA$:FORI=0TO7:B$=B$+CHR$
(VAL("&H"+MID$(A$,I*2+1,2))):NEXT:SPRITE
$(J)=B$:NEXT
20 DATA 7050F02E3966797E
30 DATA 0E0A0F649C669E7E
```

!!キーボードにふれてみよう②

```
PUTSPRITE 0,(50,100),15,0:PUTSPRITE 1,(1
50,100),15,1
```

⬆キーボードにふれてみよう①と②を実行したあとの画面。ちゃんとカルちゃんが表示されたかな。スプライトパターン0は左向き、パターン1は右向きのカルちゃんだ

⬆今月のファンダムに掲載されている「オイルショック'91」のゲーム画面。オイルの海に追い詰められていくカルちゃんの印象が強く、ついついパターンを拝借してしまった

キャラクタで画面を作る

カルちゃんがMSXに登録できたら、次はカルちゃんのために池を作ろう。

前回の講座で、キャラクタパターンの定義のしかたを教えたが、それを使って池を作るのだ。

ここでも、キャラクタパターンをちょっと拝借して使うことにした。

左の「キーボードにふれてみよう③」を打ちこんでRUNしてほしい。

おっと失敗、失敗。ちょっといっておかなくてはいけないことがあった。

「キーボードにふれてみよう①」を打ちこんだあと、この「キーボードにふれてみよう③」を打ちこむと、2つともMSXのなかに登録されるのだ。

だから、「キーボードにふれてみよう③」を打ちこみ終わったあとで、「LIST」と打ちこんで調べてみると、画面には2つとも表示されたはずだ。

だからって「どうしよう」なんて思わなくてもいい。

わたしが失敗といったのは、説明するまえに打ちこむようにいってしまったからだ。

さて、ここでRUNしてもつまらないので、つぎの「キーボードにふれてみよう④」も打ちこんでからRUNしよう。

画面が右の写真のようになったらOkだ。

画面に表示されたキャラクタの両端が池の岸で、そのあいだが池になる。

これで、カルちゃんの遊ぶ舞台ができあがったことになる。

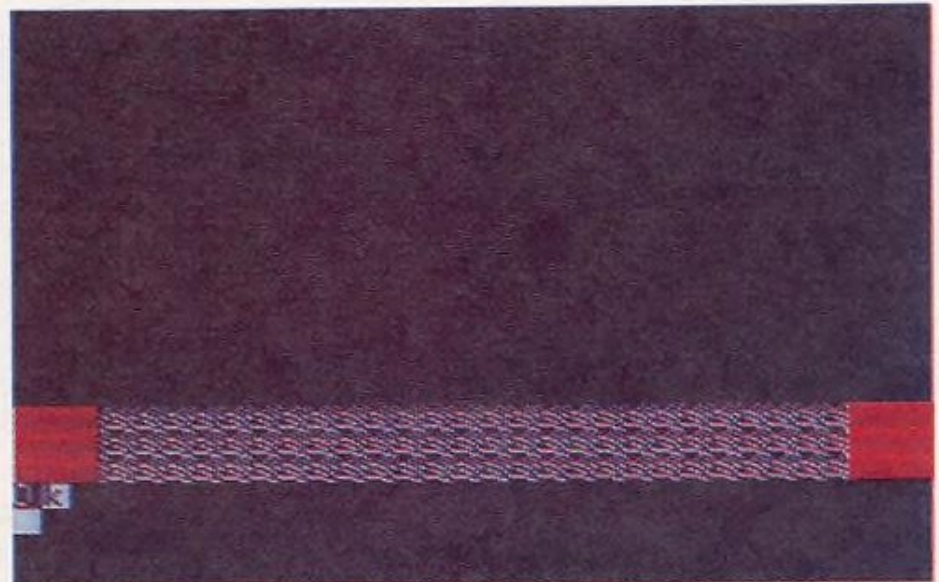
ところで、前回もすこし触れたが、スプライトとキャラクタは別の世界のものだから、表示するときの場所を指示する方法がちょっと変わってくる。

詳しくは下のカコミにあるので、読んでおこう。

!!キーボードにふれてみよう③

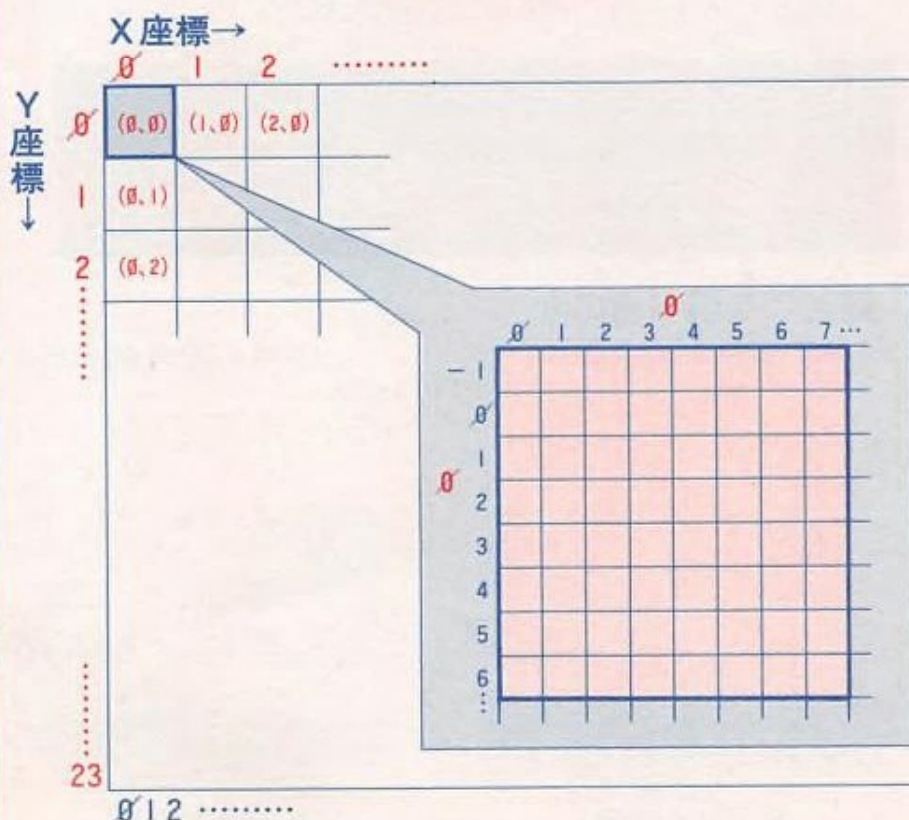
```
40 FORI=0TO1:READA$,B$,C:AD=ASC(A$)*8:FORJ=0TO7:VPOKE ASC(A$)*8+J,VAL("&H"+MID$(B$,J*2+1,2)):NEXT:VPOKE&H2000+ASC(A$)¥8,C:NEXT
50 DATA a,D669B6FFFFFFFFFF,&H68
60 DATA h,DEBD639C73BCC13E,&H4F
```

!!キーボードにふれてみよう④

```
70 KEYOFF:LOCATE0,16:FORI=0TO2:PRINT"aaa";STRING$(26,"h");"aaa";:NEXT
```

キーボードにふれてみよう③と④を打ちこんでRUNしたあとの画面。この池にカルちゃんが泳ぐことになる。行70をすこしいじくれば、ちがった感じの池も作れるぞ

SCREEN1の画面の2つの座標系

SCREEN1の画面には、2つの座標系がある。それは、キャラクタを表示するときのテキスト座標と、スプライトを表示するときのスプライト座標だ。

テキスト座標は、PRINT文で表示する文字の位置を指定するときに重要になってくるもので、たいていはLOCATEという命令を使って位置を示す。

スプライト座標は、スプライトを表示するときに重要なもので、スプライト以外では使用しない座標系なのだ。スプライトはふつうはPUTSPRITEを使って表示し、そのとき指定する座標はこのスプライト座標となる。

テキスト座標とスプライト座標は、ときたま混同してしまうが、じつはまったくちがう座標系なので注意したいところだ。

まあ、スプライト座標は1ドット単位、テキスト座標は8ドット(1文字)単位と覚えていればいい。

■スプライトとキャラクタの座標系

左の図はSCREEN1の画面の2つの座標系を表したものだが、テキスト座標に関しては、WIDTH32のモードでの数値になっている。

■テキスト座標

テキスト座標とは、文字が表示されるときの位置のことで、8ドット単位、つまり1文字ぶんで1つの座標となっている。だから文字半分だけ下にずらして表示するなんてことはできないのだ。

また、テキスト座標は基本がWIDTH32の設定になっているので、WIDTHの値が変われば、LOCATEで指定する位置もずれる。ただし、VRAMでのアドレスを計算するときは基本の座標系で計算しなければいけないので注意しよう。

■スプライト座標

スプライト座標はX座標は0から始まるが、Y座標は−1から始まっている。そのためにテキスト座標からスプライト座標を計算するときはY座標を−1する必要があるのだ。1ドット単位で指定できるので、文字よりも細かな動きができるのだ。

スプライトを動かそう

いままで打ちこんでもらった「キーボードにふれてみよう」の①から④は、前回までのおさらいのようなものだったが、いよいよこれからが本番になる。

まず､右にある「キーボードにふれてみよう⑤」を打ちこんでほしい。

もちろん、①から④のものに続けて打ちこむのだ。

さあ、これで今回のサンプルプログラムは完成となったので、さっそくRUNしてみよう。

画面に表示された池で、カルちゃんが泳いでいれば成功だ。

このカルちゃん、うまいこと岸までたどりつくと、クルッと向きを変えて泳いでいる。

注意したいのは、カルちゃんが勝手に岸と池の区別をして泳いでいるわけではなく、プログラムによってそうなるように仕組んであるということだ。

■まず、プログラムでやるべきことを考えよう

カルちゃんの移動処理では、カルちゃんの移動先のすぐ下にあるキャラクタが何かを調べ、それにより進むべきか向きを変えるかを決定しているのだ。

具体的にいうと、カルちゃんのつぎに進む先が岸だったら向きを変える。また、そうでなく池であれば、そこに進む。といった具合のものだ。

今回のカルちゃんの場合では、スプライトが8×8の2倍表示になっているので、調べなければいけない場所もそれに応じて右向きの場合と左向きの場合で変わってくるのだ。

右下の図を見てみよう。この図の左側がカルちゃんが左向きのときに、どこのキャラクタを調べればいいかを、赤で示してある。右側はカルちゃんが右向きのときのものだ。

カルちゃんを表示するのに使っているスプライト座標(X, Y)に対応するテキスト座標からみると、この調べる座標はY座標がともに＋2されている。

これはカルちゃんが池に浮いているためで、もしカルちゃんが水中にいるなら、Y座標の修正値も変わってくるのだ。

また、X座標は左向きのとき－1で、右向きなら＋2になっている。

この場合は、カルちゃんのスプライトの大きさが関係していて、右向きと左向きとで、数値の大きさが違うのだ。

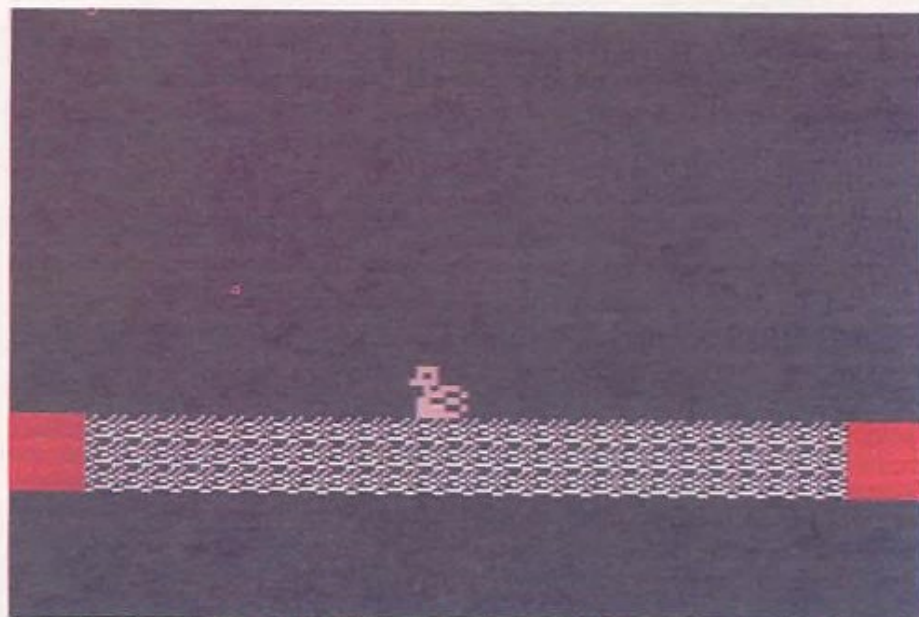
■調べる座標がわかったら、VPEEKで調べてみよう

さて、調べる座標がわかったら、こんどはどうやって調べるかが問題になる。

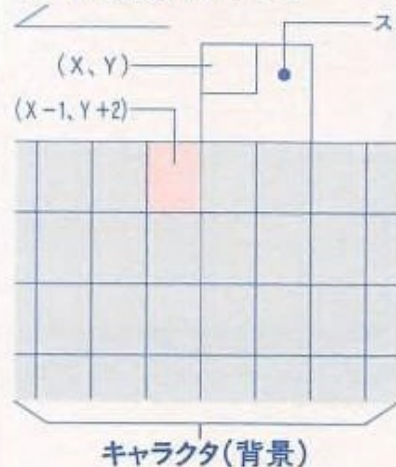
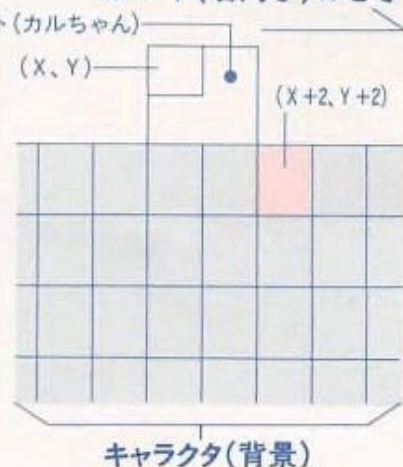
VRAMを直接調べると、ここではひとことでいっておくが、このVRAMというヤツは、最初のうちはなかなかなじめないもので、いくら理屈を聞いてもよく理解できないものだ。

でも、使っていなくては話にならないので、来月の講座では、SCREEN1のVRAMに思いきり迫ってみよう。

‼キーボードにふれてみよう⑤

```
80 X=15:Y=14:S=0:AD=&H1800+(Y+2)*32
90 PUTSPRITE 0,(X*8,Y*8-1),10,S
100 IFS=0THENIFVPEEK(AD+X-1)=ASC("a")THENS=1:GOTO120ELSEX=X-1
110 IFS=1THENIFVPEEK(AD+X+2)=ASC("a")THENS=0ELSEX=X+1
120 FORI=0TO100:NEXT:GOTO90
```

●キーボードにふれてみよう⑤を打ちこみ終わり、完成したサンプルをRUNしたあとの画面。カルちゃんは元気よく池を泳いでいる

●カルちゃんは岸までたどりつくと、クルッと向きを変えて、また泳いでいく。いっけんかんたんそうに見えても、ここにスプライトの妙味があるのだ

■どこを調べるのか

今月の講座はちょっと増ページして、SCREEN1のVRAMにせまってみた。VRAMは魅力たっぷりの宝箱だから、とにかく使ってモノにしよう。

VRAMの魅力

SB講座の読者のなかには、VRAMといわれてもピンとこない人も多いだろう。

かんたんにいってしまえば、VRAMとは画面表示に関係するデータのしまわれているところのことだ。

でも、それだけじゃVRAMがなんだかよくわからない。

「わからない」という気持ちはやっかいなもので、いったんそう思いこんでしまうと、なかなか理解できなくなってしまう。

そこで今回の講座では、VRAMの仕組みと魅力を理解するための手助けをしよう。

■VRAMという宝箱をちょっとのぞいてみよう

まずは難しいことはぬきにして、VRAMの楽しそうな雰囲気をつかんでほしい。

ではさっそくMSXの電源を入れて、右のキーボードにふれてみよう①を実行しよう。

画面にはなにも変化はないが、ちょっとスプライトを表示する命令を打ちこんでみよう。

ここで、「スプライトパターンを定義しなくていいの?」と思った人はなかなか冴えてるぞ。

ふつうならスプライトパターンを定義しないとスプライトは表示されないところなんだが、さっき打ちこんだキーボードにふれてみよう①で、MSXの文字とおなじ形のスプライトパターンが定義されているのだ。

つぎにキーボードにふれてみよう②を打ちこんでみよう。

画面にちらっとスプライトが表示されただろう。

カーソルを画面の上に移動して、そのへんに文字を打ちこんでみよう。

画面のところどころにスプライトが現れただろう。

じつは、PUTSPRITE命令を使わなくても、スプライトを表示できるのだ。

VRAMにはスプライトの色や位置、どんなパターンを表示するかなんていう情報がしまわれている部分があり、画面に文字を表示すると、その部分のVRAMが変化してスプライトが表示されるのだ。

こんな遊びができるのも、VRAMの魅力のひとつだろう。

■VRAMの楽しさはまだまだたくさんある

ファンダムに掲載されているプログラムには、よくVPOKEとかVPEEKといった命令をみかける。

そういったプログラムでは、みんなVRAMの良さを利用しているのだ。

いろんな形の色つきの文字できれいにほどこされたゲーム画面なんかはまさにそう。

VRAMの魅力は数え上げたらきりがないのだ。

‼キーボードにふれてみよう①

```
BASE(9)=BASE(7)
```

```
MSX BASIC version 4.0
Copyright 1990 by Microsoft
23414 Bytes free
Disk BASIC version 1.0
Ok
SCREEN,1:BASE(9)=BASE(7)
Ok
PUTSPRITE 0,(100,100),8,1
Ok

color auto goto list run
```

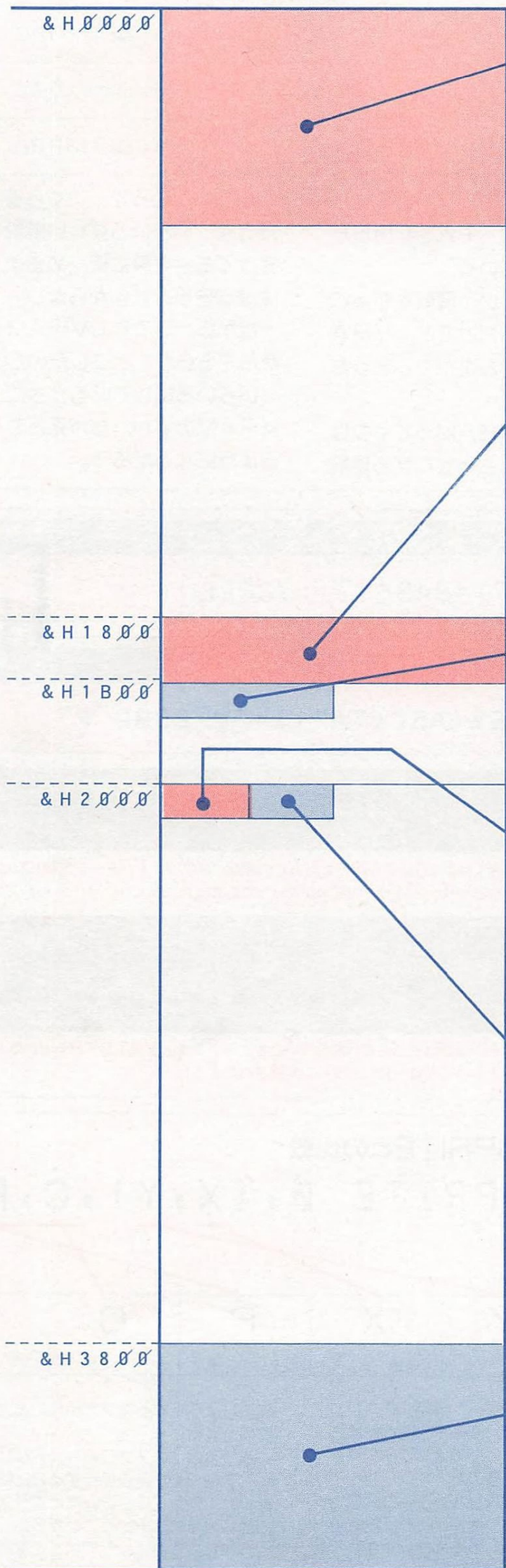
↑キーボードにふれてみよう①を実行したあと、PUTSPRITE命令で適当にスプライトを表示させてみる。スプライトパターンは定義されてないはずなのに……⁉

‼キーボードにふれてみよう②

```
BASE(8)=BASE(5)
```

↑さらにキーボードにふれてみよう②を実行すると、画面にはスプライトがウジャウジャ表示されてしまった。カーソルを上に移動して文字を打ちこむとスプライトが反応する

SCREEN1のVRAM

VRAMマップ

SCREEN1のVRAMマップ。空きエリアがたくさんある

各エリアの解説

■キャラクタパターンジェネレータテーブル(&HØØØØ~&HØ7FF)

●内容
文字(キャラクタ)の形の情報が格納されている。SCREEN命令で画面をSCREEN1にしたとき、保存されているデータが初期化される。

●アドレス
ここの領域の開始アドレスはBASE(7)の位置。ふつうは0になっている。データはコード順に1文字につき8バイトずつ、全部で2048バイトの大きさがあるので、アドレスの計算方法はその文字のコード×8となる。ただし、VRAMではキャラクタコードとは少しちがったコードで文字の管理をしていて、ふつうの文字はキャラクタコードとおなじだが、グラフィックキャラクタは0~31のコードで管理されている。これはVRAMでは共通のコードなので覚えておこう。

■パターン名称テーブル(&H18ØØ~&H1AFF)

●内容
画面のどこに、どんな文字を表示しているかの情報が格納されている。CLSなどで画面を消そうとすると、ここの内容がすべて空白に書き変わり画面が消える。ここの情報が書き変われば、画面もそれに応じて変化していくのだ。ここはスプライトと文字の衝突判定に使用されることが多く、慣れるとよく使うところ。

●アドレス
BASE(5)のアドレスからこの領域で使用される。領域の大きさは、横32×縦24の768バイトで、各アドレスに対応する画面の位置は、画面のいちばん左上から横に1文字につき1バイトずつ、いちばん右までいくと次の行のいちばん左に移動し、最後がいちばん右下になる。ここではWIDTHの値に関わらず、横の文字数はつねに32、縦の行数は24に固定されている。だから、LOCATEØ,Øの位置が、必ずしもこの領域の始めになるとは限らないので注意が必要。

■スプライトアトリビュートテーブル(&H1BØØ~&H1B7F)

●内容
ここにはスプライトの表示に関する情報がまとめられている。スプライトは基本的にスプライト面で管理されていて、面ごとに表示するスプライトの座標、色、パターン番号が格納されている。

●アドレス
この領域の開始アドレスはBASE(8)の値から。スプライト面は0~31の全部で32面あり、1面につき表示するスプライトのY座標、X座標、色、パターン番号の合計4バイトが使用される。だから領域の大きさは128バイトとなる。ファンダムではしばしば使われているが、直接ここを使うより、PUTSPRITE命令を使うほうがわかりやすい。

■カラーテーブル(&H2ØØØ~&H2Ø1F)

●内容
SCREEN1の画面で表示される文字の色情報が格納されている。中途半端な指定方式で、コード順に8文字につき2色(文字の色と背景の色)を指定する。COLOR文で画面の色を変えると初期化される。ファンダムではとてもよく使われている領域だ。

●アドレス
BASE(6)のアドレスから使用される。256種類の文字の8文字単位の色指定なので、領域の大きさは32バイトとなる。格納されるデータは、前景色(文字の色)×16+背景色となっている。てっとりばやく効果的な装飾をほどこせるので、使用頻度は高いところだ。

■パレットテーブル(&H2Ø2Ø~&H2Ø3F)

●内容
MSXでは、0~15のカラーコード(パレットコード)が使えるが、そのパレットの赤、青、緑の各成分の情報が保存されているところ。通常はCOLOR=(パレット番号,赤の輝度,緑の輝度,青の輝度)でパレットを切り換える。COLOR=RESTOREにより、ここの情報がパレットに切り換えられ、それまではここの値を変えてもパレットは切り換わらない。SCREEN命令かCOLOR=NEWの実行で初期化される。

●アドレス
アドレスは&H2Ø2Øから始まり、各パレットにつき2バイトずつ、計32バイトの領域を持つ。めったに使わない。

■スプライトパターンジェネレータテーブル(&H38ØØ~&H3FFF)

●内容
スプライトパターンの保存場所。スプライトパターンの登録は8×8ドットのパターンが256枚分できるが、領域の大きさが固定されているので、16×16ドットのスプライトでは64枚しか登録できない。SCREEN命令を実行したときの、第2パラメータにより初期化される。

●アドレス
BASE(9)の位置からキャラクタパターンジェネレータテーブルとおなじように2048バイトの領域がある。両者の働きはよく似ていて、文字の形かスプライトの形かの違いにすぎない。ただ、文字の形は定義用の命令はなく、スプライトにはSPRITE$が用意されている。

どんどん使っておぼえよう

P49の図はマニュアルなどでわりとよく目にするVRAMマップというものだ。

これは、初期状態のSCREEN1のVRAMが、どんな目的のために、どのように使われているかを表したものだ。

はじめのうちはかなり抵抗のある図だが、慣れてくれば必要なものになってくる。

VRAMの各領域に対する解説には、ひととおり目を通してもらえただろうか。

けっこう硬めの解説だったので、よくわからないという人も多いだろう。

ここでは、各エリアにもう少し踏みこんでみようと思う。

■BASE関数を使ってVRAMで遊ぼう

右の表は各エリアの初期アドレスを並べたものだが、アドレスの値といっしょにBASEなるものが書かれている。

このBASEという関数は、ほとんどのマニュアルでは説明を省略されているもので、引数によりVRAMの各テーブルのアドレスを教えてくれる。

また、これに数値を設定すれば、つぎからはその数値の場所がアドレスとして使用される。

■SCREEN1の各エリアの初期アドレス

エリアの名称	初期アドレス
キャラクタパターンジェネレータテーブル	BASE(7)=&HH0000
パターン名称テーブル	BASE(5)=&HH1800
カラーテーブル	BASE(6)=&HH2000
スプライトアトリビュートテーブル	BASE(8)=&HH1B00
スプライトパターンジェネレータテーブル	BASE(9)=&HH3800

下にある各エリアの少し詳しめの解説では、それを利用した遊びを試みている。

BASEという関数をちょこっといじくるだけでも、VRAMはじゅうぶん遊べてしまうものなのだ。

まったくVRAMってやつは使っていなくちゃぜんぜん理解できないところで、いくら役割だとか、いろいろ詳しい解説を受けても、実際に確かめるまでははっきりとはわからない。

だから、ここでもVRAMで遊んでもらうことにしたのだ。

いろいろ試しているうちに、VRAMの姿かたちが見えてくるようになるだろう。

2つのジェネレータテーブル

ここではキャラクタパターンジェネレータテーブルとスプライトパターンジェネレータテーブルについて、少し踏みいってみよう。

始めのページにあるキーボードにふれてみよう①でも登場した、
BASE(9)=BASE(7)
という命令には、いったいどんな意味があるのだろう。

ここでも、それを利用した遊びをもうひとつ紹介しよう。

始めのページでは、スプライトパターン定義を一瞬にしてやってしまうというもので、スプライトの表示にまで触れていた。

こんどはキャラクタの側でのメリットを見てみることにするのだ。

右のキーボードにふれてみよう③と④を実行しよう。

なんと、スプライトパターン定義に使うSPRITE$という関数で、スプライトパターンを定義したら、キャラクタパターンの定義までされてしまったのだ。

これはじつにおもしろい遊びで、じゅうぶん実用性があるのだ。

ふつうでは、キャラクタパターン定義は、VPOKEを使って直接VRAMを書き換えるしかなかったが、このように関数で定義できるので便利だし、スプライトとキャラクタでおなじパターンを使うときなど、一回の命令実行で両方のパターン定義ができる。

こんなことができる理由も、先に示したBASE関数のしわざなのだ。
BASE(9)=BASE(7)
の実行で、キャラクタパターンジェネレータテーブルとスプライトパターンジェネレータテーブルがおなじ位置に設定されたことになり、VRAMの内容が変われば、両方のジェネレータテーブルに影響が出てくるのだ。

2つのジェネレータテーブルは遊びがいがあるので、いろいろ試そう。

‼キーボードにふれてみよう③

```
BASE(9)=BASE(7):SCREEN1
```

‼キーボードにふれてみよう④

```
SPRITE$(ASC("A"))="♥~BBBB~♥"
```

```
SPRITE$(□SC("□"))="♥~BBBB~♥"
Ok
```

↑キーボードにふれてみよう③と④を実行したあとの画面。スプライトパターン定義したはずなのに、なぜかアルファベットの「A」の文字が変わってしまった

```
SPRITE$(□SC("□"))="♥~BBBB~♥"
Ok
PUTSPRITE 0,(100,100),8,65
Ok
□□
```

↑スプライトパターン定義はきちんと行われていたが、スプライトを表示してみると、きちんと行われていた。ということは一緒にパターン定義されたようだ

スプライトアトリビュートテーブル

スプライトアトリビュートテーブルとは、スプライトの表示に関するデータの保存されているところだ。

スプライトはスプライト面というもので管理されていて、1つの面に表示できるスプライトは1枚に限られていたり、スプライトが重なったとき、どちらが優先されて表示するかも、この面番号に関係してくる。

スプライト面番号は0～31まであり、画面にはスプライトは32枚までしか表示できない仕組みだ。

ここでは、このテーブルの詳細を説明することにしよう。

開始アドレスはふつうは&H1B00からになっていて、BASE(8)の値によって変化する。

データは各スプライト面ごとに4バイトずつ、続けて保存されている。

1バイト目は表示するスプライトのY座標が入るが、格納されるデータはスプライト座標となる。

また、ここの値が209になるとそのスプライトは表示されず、208ではこのスプライト面以降のスプライトは表示されないので注意が必要だ。

2バイト目は表示するスプライトのX座標が入る。

3バイト目は表示するスプライトのパターン番号が入る。

このパターン番号は、スプライトモードが8×8のときはそのままの番号が入るが、16×16のモードのときは、連続した4つのパターンで1つのスプライトとみなした数値が入る。

かんたんにいえば、0～3のパターンを合わせてパターン番号0のスプライトとするので、0～3のどれを書きこまれてもおなじ形のスプライトパターンが表示されるのだ。

最後の4バイト目はカラーコードでふつうは0～15の値が入るが、128を足した値が入ると、X座標が-32されてスプライトが表示される。

Y座標、X座標で指定する座標は、表示しようとするスプライトの左上の座標となる。

ふつうはこのスプライトアトリビュートテーブルを直接操作することは少なく、1画面プログラムなどでリストの節約のために用いることが多い。

しかし、場合によってはここを操作してとても楽しいことができるので、いつかこのページで紹介しよう。

■PUTSPRITEとの比較

PUTSPRITE 0,(X,Y),C,P

VRAMで遊ぶときの注意

ここまでSCREEN1のVRAMについてひととおり触れてきたが、ここにVRAMを使うときの注意をまとめておく。

今回の講座では、BASE関数の内容を変えていろいろ遊んでみたが、遊び終わったらリセットしたほうがいいだろう。

たとえば、キーボードにふれてみようの①か③を実行したあとで、SCREEN, 1などと実行すると、画面はとんでもないことになってしまう。

このときは、SCREEN1をもういちど実行すればもとにもどるが、キャラクタパターンやスプライトパターンは初期化されてしまうので注意。

VRAMに一生懸命データをセットしたあとでの注意として、SCREEN、COLOR、CLS、WIDTHなどの命令を使うときには注意してほしい。

これらの命令は、少なからずVRAMに影響を与えるもので、たとえばCOLOR命令を使うと、カラーテーブルが初期化されてしまうのだ。

プログラムでVRAMを操作するときは、画面に関係した初期設定を行ってからでないと、このような悲劇がうまれる。

■VRAMの更なる魅力

SCREEN1のVRAMを操作して遊ぶ。

これはかなり楽しい作業で、わかればわかるほどハマっていくものだ。

しかし、ここにあげたいくつかの例に止まらず、VRAMにはまだまだたくさんの遊びが隠されている。

下にあるパターン名称テーブルのカコミでもちょっと紹介した、ページ切り換えというテクニックや、最近、また質問が増えてきた多色刷りモードなど、いったいいくつあるのだろう。

数えるのが面倒なので、数えないが、VRAMだけではこれらのテクニックは使えないとだけいっておこう。

べつにテクニックの出し惜しみをしているわけではなく、このあたりの話になってくると、いろいろと予備知識も必要になってくるし、スーパービギナーの域を超えてしまうからだ。

VRAMは理解しておき、何かのときに利用するところだと思うようにしよう。

さて、今回までずっとVRAMに関することで話を進めてきたが、次回からは少し視野を変えて、プログラム的な部分にせまってみよう。

パターン名称テーブル

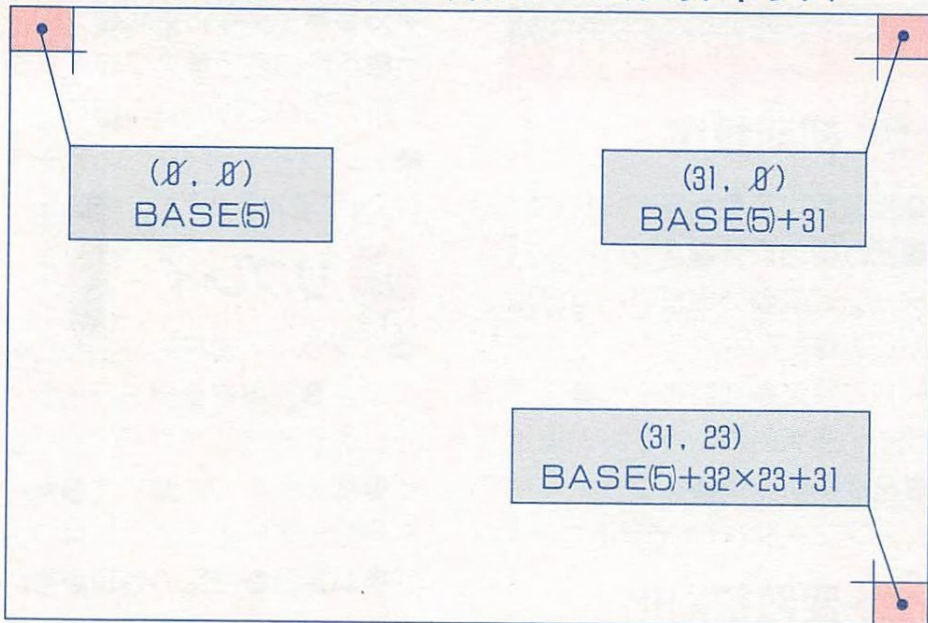
SCREEN1の画面では、横の文字数は最大で32文字、縦の行数はファンクションキーの内容を表示する行を含めて24行まで使える。

この32×24の文字を表示する画面のことをパターン名称テーブルと呼び、VRAMのパターン名称テーブルは実際にデータの書きこまれる画面ということで、アクティブページと呼ばれる。

実際にデータの書きこまれる、という意味は、カーソルのあるところに、文字を書きこめるということだ。

わざわざパターン名称テーブルという名前があるのに、アクティブページなどという呼ばれ方までしているからには、当然、理由がある。

パターン名称テーブルには、表示と読み書きの2つの画面があるのだ。

■SCREEN1の画面と名称テーブルのアドレス

■名称テーブルのアドレス計算方法

BASE(5)+Y座標×32+X座標

ここで扱うのは、読み書き専門で、内容を変えたりできるが、表示のほうはただ表示するだけだ。

なぜこのようにわかれているか、なんてことはわたしの知るところではないが、じつはこのオカゲで楽しいことができるのだ。

ページ切り換えと呼ばれているテクニックがそれで、かんたんにいえば、紙芝居のようにVRAMにいくつかのパターン名称テーブルを作っておき、それをあとから順番に表示してアニメーションさせるものだ。

P49で紹介したVRAMマップを見てみると、かなりの部分が使われていないのがわかる。

この空いている部分にいくつかパターン名称テーブルを作り、アニメーションさせるのだ。

詳しいやり方はここでは紹介しないが、機会があったらいつか紹介しよう。

ところでパターン名称テーブルにはまだいくつかの利用法がある。

その中で使用頻度の高い、スプライトと文字との衝突判定をするには、このテーブルの存在は不可欠なのだ。

この方法を使うためには、スプライト座標とテキスト座標を、正しく理解していないとうまくいかないが、試しながらやって覚えるのもいいだろう。

パターン名称テーブルのアドレスの計算方法は、上にあるように、BASE(5)+Y座標×32+X座標となっていて、この座標はWIDTHの値に関係しない決まった座標系のものだ。

左にある図は、SCREEN1のテキスト座標系とアドレスとの対応を表している。

ちょっとHOMEキーを押して、カーソルをいちばん左上にしよう。

カーソルを移動させたら、何かの文字キーを押しっぱなしにしてほしい。

ずうっと文字が表示されていくが、パターン名称テーブルはちょうどこれと比例してアドレスが変化するのだ。

最後に、

WIDTH32:KEYOFF

を実行して、LOCATE31, 23の位置、つまりいちばん右下に文字をPRINT文で表示させてみよう。

結果はうまくいかない。

ここにはVPOKEで直接パターン名称テーブルを書き換えないと、文字は表示できないのだ。

カラーテーブルとパレットテーブル

ここでは色に関する2つのテーブルをまとめて紹介しよう。

カラーテーブルは慣れるとしょっちゅう使うところだが、パレットテーブルはほとんど使わない。

しかし、覚えておいても損ではないので、よく読んで理解しよう。

■カラーテーブル

カラーテーブルとはSCREEN1の画面に表示される文字に、8文字ごとの色指定ができるもの。

8文字ごとの色指定といっても、自由な組み合わせではなく、8文字のセットは決まっていて、キャラクタコードの順番に8文字単位となる。

また、指定はVRAMでの文字コード順になっているので、グラフィックキャラクタの色指定からとなる。

アドレスの計算方法は、

BASE(6)+その文字のVRAMでの文字コード¥8

となる(ここで使っている¥マークは計算記号で、割り算と似た意味を持つ)。

カラーテーブルは、ちょっと変えるだけでかなりの効果があるので、はやく使いこなせるようになろう。

■パレットテーブル

パレットテーブルは、現在のカラーパレットの内容の保存場所のことだ。

これは、SCREEN1の画面でもMSX1にはないもので、COLOR命令を使ってパレットを切り換えるとここに情報が保存される。

ほとんど使うことはないが、画面をBSAVEで保存したときのパレットの切り換えなどで利用する。

パレットの切り換えのしかたはCOLOR=RESTOREで行う。

■カラーテーブルの内容

8文字単位の色データ(前景色×16+背景色)

■パレットテーブルの内容

2バイトで1セット(1バイト目……赤成分×16+青成分／2バイト目……緑成分)

〈ファンダムニュース／読者アンケート4月号結果発表〉4月号のファンダムプログラム、BEST5! 5位は穴に落ちないように左へ進んでいく『SKELETON』。4位はペットを助けに落ちていく『PET FALLII』。同じ色のブロックを重ねたらゲームオーバーとなる『SUBMERGE』が3位。2位にはコンピュータ相手にカードゲームで熱中する『SIEGE』。そして1位はアクション&パズルゲームの『タコ&イカ』だ。

スーパー ビギナーズ

超初心者 Super Beginners' 講座 第8回

VRAMはなんだかんだいってもしょせんはプログラムで利用していくところ。これからのSB講座では、プログラムの処理作りに目を向けていこうと思っている。

ビギナー向きのテクニック:パターン切り換え

前回の講座はちょっと詰めこみすぎたかなと、少し反省しているが、VRAMのすべてを急いで理解する必要はない。

今後、このスーパービギナーズ講座では、プログラムのほうに目を向けて行こうと思っているが、その中で少しずつVRAMを利用していく予定なので、前回のVRAMマップと各エリアの説明を資料として使ってもらえれば光栄である。

いまはVRAMのことはさっぱりわからなくても、そうしているうちに自然に体で覚えてしまうはずだ。

では、今月の講座を始める。

■スプライトパターン切り換えはビギナー向きのテクニック

スプライトパターンの切り換えテクニックとは、いったいどんなものだろうか。表示されているスプライトをパタパタ切り換えて、アニメーションしているように見せることなのだが、はっきり決まったやり方がない。そんな判然としないスプライトパターン切り換えが、どうして初心者向けなのだろうか。

それは、このテクニックに必要なデータと知識がかんたんだから、というのが理由だ。

■スプライトパターンとスプライトを表示する命令

ここでいうデータとはスプライトパターンのことで、切り換えるためには2枚以上のパターンが必要になる。そして、それを表示する命令と変数を使ったかんたんな足算や引算などの計算をする知識があれば、じゅうぶんできるテクニックなのだ。

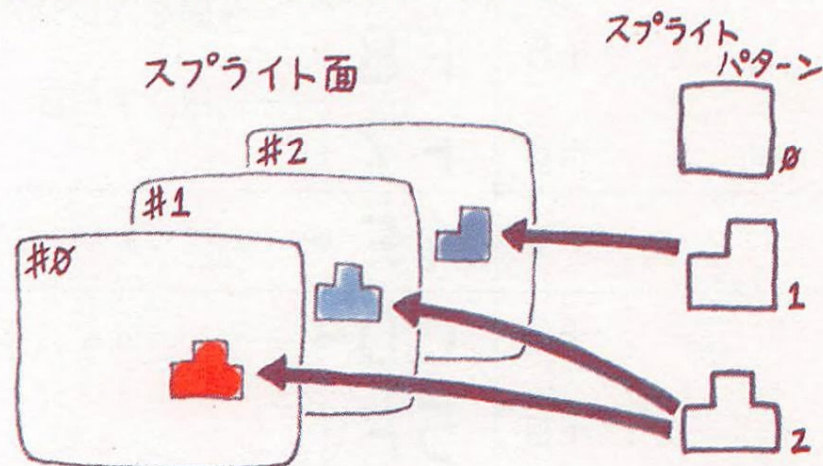
まずはどんなアクションをそのスプライトにさせるかを考えよう。例えばくるくる回るコインの表現や、颯爽(さっそう)と走る馬、スカイダイビングのパラシュートなど。自分のやりたいイメージがわいたら、それをスプライトパターンで表現するのが最初の仕事だ。パラシュートのたたまれた状態や出た瞬間、開いたところなどをスプライトパターンで作るのだ。それができたらちょっとスプライトの表示を下のカコミでおさらいしておこう。

スプライトの表示:PUTSPRITE

PUTSPRITEの働きを説明すると、「そのスプライト面のどこに、何色で、どのパターンを表示するか」となる。スプライトパターンはスプライト面に置かれてはじめて画面に現れ、そのスプライト面の色を変えると、指定された色でスプライトが表示される。

スプライト面とは、文字やグラフィックが表示される画面の上にあり、スプライトを1つ表示することができる画面のことで、0～31番の合計32枚がある。面番号0がいちばん手前にあり、面番号31のつぎに文字やグラフィックが表示される画面がある。だからスプライトと文字が重なると、かならず文字が隠れるのだ。

スプライトパターン切り換えは表示するスプライト面番号は変えずに、表示するスプライトパターン番号のみを変えて、スプライトの表示をアニメーションのように変化させるテクニックなのだ。

■スプライトの表示されるしくみ

```
PUTSPRITE 0,(100,100),15,0
```

スプライト面番号
どのスプライト面に指示を与えるかを0～31の番号で指定。省略はもちろんできない。

スプライトの表示座標
どこにスプライトを表示するかスプライト座標で指定。Y座標はスプライトの消去の指定にも使用。

スプライト面の色
そのスプライト面に表示するスプライトの色を指定。変更しない場合は省略することもできる。

スプライトパターン番号
そのスプライト面に表示するパターンを番号で指定。省略した場合はそれまでと同じパターンを表示。

※スーパービギナーズ講座では今後の参考のため、ご意見、ご感想、ご質問などを募集します。また、初心者の方へのメッセージなども受け付けます。応募は必ずハガキで、MSX・FAN編集部「SB講座」係まで(住所は76ページ)。この記事で使用したハガキの投稿者には、Mファン特製テレカをさしあげます。

採用作品での実用例

■切り換えとは変数の値を切り換えるための計算のこと

さて、切り換えに使うスプライトパターンが出来上がり、そのスプライトを表示する方法がわかったら、いよいよスプライトパターン切り換えの仕組みにせまってみよう。

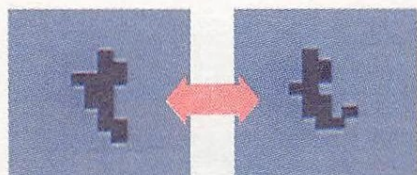
ところでスプライトパターン切り換えの「切り換え」とはどういうことだろうか。じつはこの切り換えというのは、変数の内容を切り換えるという意味だ。

たとえば、表示するスプライトパターン番号を変数で指定したとき、その変数がスプライトを表示するたびに1と0の値で切り換わるとしたら、これはもう立派なスプライトパターン切り換えになるのだ。なぜなら、変数の内容が変われば、その変数で指定・表示されるスプライトパターンも変わるからだ。

このことから、スプライトパターン切り換えのつめの作業は表示パターン番号の指定に使う変数の用意と、その変数をどのように切り換えるかにある。

しかし前にも述べたが、切り換えには計算やデータ、IF文の使用などいろいろあり、決まったパターンが存在しない。そこで今月のファンダムから、それぞれパターンの違う4つの作品を実用例として下に掲載した。各作品でのスプライトパターン切り換え処理の部分を説明しているので、読んでほしい。

『アシカの修業』の場合

52ページの『アシカの修業』では、アシカの動きにスプライトパターン切り換えを使っている。下のリストの赤い部分が切り換えに関するところで、行4でスプライトパターンを指定する変数Jを切り換えていて、ゲーム中つねに実行されているので、変数Jの値は変わり続ける。計算の内容は、J＋1を行い、それを「MOD4」した結果をまた変数Jにもどしている。「MOD」というのは計算記号で、割り算を行いその余りを計算結果とするものだ。そして、行5にあるPUTSPRITEでスプライトを表示するとき、変数Jを使って表示するスプライトパターン番号を指定しているのだ。実際にはJ／2なので、2回に1回の割合で、表示されるスプライトパターンが切り換わるのだ。

```
4 J=(J+1)MOD4:SPRITEON:X=X+A:Y=Y+B:IFB>1
6THEN3ELSEB=B+1:A=A*(1+(X>240ORX<0)*2)
5 S=STICK(0)+STICK(1):V=V+(S=7ANDV>13)-(
S=3ANDV<58):PUTSPRITE0,(V*4,110),1,J/2:P
UTSPRITE1,(X,Y),C+6,2:IFC=6THEN3ELSE4
```

『ぷよよん』の場合

54ページの『ぷよよん』では、ふだんはスプライトパターン切り換えを行っていない。しかし、床に衝突したときに右の写真のように変化するのだ。スプライトパターン番号の指定には変数Bが使われていて、行3のはじめて前もって0にしている。そして、プレイヤーが動いたあと、その移動先にある文字がどんなものかを判定して、床になっていれば変数Bを1に変更している。つまり文字をスプライトの判定によって切り換えを行っているのだ。判定に使うIF文については今後の講座で取り扱う予定。

```
3 B=0:S=STICK(0):X=X+((S=7ANDX>0)-(S=3AN
DX<247))*3:Y=Y+W:W=W+Q:W=W+SGN(W)*(ABS(W
)>5):E=6144+(X+4)¥8+((Y+4)¥8)*32:V=VPEEK
(E):IFV=219THENW=-W:Y=Y+W:B=1ELSEIFV=131
THENQ=-Q:VPOKEE,32:F=F+1:G=G+1+H:PLAY"O6
FFA4":IFFMOD5=0THENPLAY"CDECDEF4":GOTO2
5 N=N+O:M=M+P:PUTSPRITE1,(X,Y-1),15,1+B:
PUTSPRITE0,(N,M-1),C,0:IFABS(X-N)>6ORABS
(Y-M)>6THENIFY>-50ANDY<230THEN3ELSE7
```

『ぐりぐりSLIME』の場合

56ページの『ぐりぐりSLIME』では、スライムの向きによって表示されるスプライトのパターンも変わるという変則的な切り換えを使用している。スプライトパターン番号を指定しているのは、配列変数A(I)で、スライムの方向として使われている。また、基本的には配列でなくても構わない。ここでは、スティックの入力によって、配列変数Aの値を1増やすか減らすかを決めているが、その計算部分は関係演算と呼ばれるものを使っている。関係演算というのは、「A＝5」や「D＞0」などといった、2つの値を比べ、「＝」や「＞」で示した関係が成り立つなら値が－1になるというものだ。関係演算とスティック入力も今後の講座で必ず取り上げるくらい大切なものなので、マニュアルで調べておこう。

```
50 C=C+1:FORI=0TO1:D=I*3+5:S=STICK(I):G=
STRIG(I):A(I)=A(I)+(S=7)-(S=3):A(I)=A(I)
*-(A(I)<12):IFA(I)<0THENA(I)=11
90 PUTSPRITEI*2,(X(I),Y(I)-1),15,A(I):PU
TSPRITEI*2+1,(X(I),Y(I)-1),D,12:NEXT:IFC
<20THEN50ELSEC=0:IFH(0)=H(1)THEN50ELSEB=
-(H(0)>H(1)):H(B)=H(B)+1:GOSUB100:GOTO50
```

『中西の挑戦』の場合

57ページにある『中西の挑戦』では、かなり特殊なスプライトパターン切り換えを行っている。それは、ふだんは表示されているスプライトに変化はないのだが、スティックの下が入力されていると、スプライトパターン切り換えが行われるようになるのだ。プログラムを見ると、かなり情況によっての処理が入り乱れているため、混乱するかもしれないので、考え方のみ説明しよう。このプログラムでは、基本的にスプライトはパターン0を表示している。ところが、移動のスティック入力のところで下の入力があると、スプライトがまるで回転しているかのように切り換わる。下の写真は切り換わっているスプライトパターンなのだが、左のパターンから右へ切り換わり、右のパターンからまた左のパターンに変化するのだ。他のプログラムでのスプライトパターン切り換えとはその点がちがう。

```
60 IFS=5ANDY(I)=103THENFORJ=103TO119:PUT
SPRITE4+I,(X(I),J),14+I,VAL(MID$("0121",
JMOD4+1,1)):NEXT:Y(I)=119ELSEIFY(I)=119A
NDGTHENFORJ=119TO103STEP-1:PUTSPRITE4+I,
(X(I),J),14+I,VAL(MID$("0121",JMOD4+1,1)
):NEXT:Y(I)=103
```

超初心者

Super Beginners' 講座 第9回

スーパー ビギナーズ

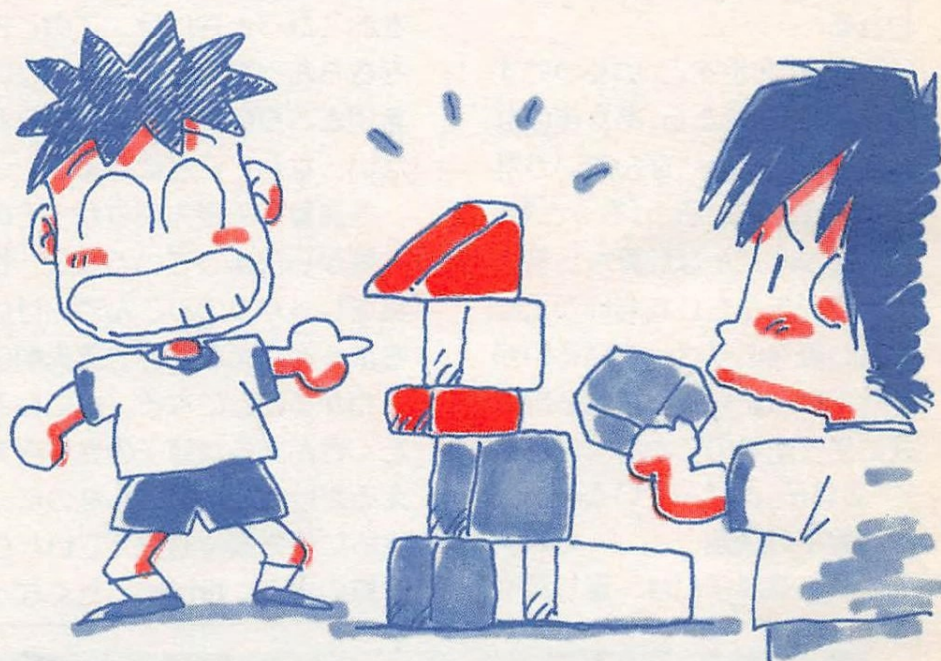
時代劇で、いともかんたんに切り捨てられたり、ゴジラの足もとで逃げまどう脇役の人たち。そんなイメージのある、プログラムの世界での脇役にせまってみよう

プログラムの影の主役たち

「ええい、ひかえおろう／」でおなじみの水戸黄門。主役というイメージにはピッタリくるものがあるが、そんな黄門さまでも、印篭を見せつけられてひかえてくれる、脇役の人たちがいるからこそ、引き立って見えるのだ。プログラムの世界にも、そういった関係があてはまるようで、BASICの命令を主役とすると、変数を使った計算式などがその脇役にあたるのではないだろうか。

■主役逆転

今回のSB講座では、この脇役的な存在の、変数を使った計算式に目を向けてみようと思う。BASICの命令を積木にたとえると、変数や計算などの要素は、BASICで組まれた積木の肉付けをする粘土のようなもので、この粘土による肉付けがうまくいけば、プログラムはどんどんよくなっていく。

たとえば、前回の講座では、パターン切り換えの方法というのをやったが、変数や計算式の、ちょっとしたくふうでできるものだった。わざわざ人気のある主役を使わなくても、きちっと脇役を押さえておけば、いい作品が出来上がるというわけだ。そう考えていくと、脇役的な存在の変数や計算こそ、プログラムの世界の影の主役といえるのではないだろうか。

BASICの命令をよく知らなくても、マシン語なんかぜんぜんわかんなくても、変数の使い方や計算のくふうがあれば、いいプログラムを作っていくことは可能なのである。

先月のおさらい:変数の切り換え

前回の講座では、ビギナー向きのテクニックとして、スプライトパターン切り換えを紹介した。

そして、そのスプライトパターン切り換えの実例として、ファンダム掲載プログラムのなかから4本を選び、スプライトパターン切り換えをおこなっている処理について解説した。どのプログラムも、スプライトパターン切り換えは変数の値を切り換えることによって実現されていたが、実際にはどのようにして変数を切り換えているのだろうか。

右にはファンダムで使用されることが多い切り換え法を4つ紹介しているが、計算式や命令を使って切り換えている。命令を使った場合と計算式を使った場合とでは、どんな違いがあるのだろうか。

IF文やFOR～NEXTを使った切り換えの利点は、どのように変数が切り換わるかがわかりやすいということだ。しかしその反面、処理が長いなどの問題もあり、またIF文などは、THEN以降とELSE以降とで、処理が2つにわかれてしまう。ところが、関係演算などの演算を使った変数の切り換えでは、このようなことはなくとても実用的だ。ただ、演算を使った切り換えはその仕組みがわかりにくいので、各演算の働きを理解していないとモノにできない。変数の切り換え法をマスターするには、BASICの基本ともいえる演算をきちんと理解しておかなくてはならないことになる。

変数Aを0と1で切り換える

●IF文を使った場合

```
IF A=0 THEN A=1 ELSE A=0
```

IF文を使った変数の切り換えは、変数の現在の値を判定し、判定結果によって変数を再設定する。Aが0なら1にし、それ以外ならAは0になる。

●ループを使った場合

```
FOR A=0 TO 1
```

FOR～NEXTループを使うときは、切り換えたい変数をループ用にし、ループを終了したら、またFORの前に処理を移して切り換えをする。

●関係演算を使った場合

```
A=-(A=0)
```

関係演算を使うと短くなる。この場合、Aが0なら(A=0)の関係が成り立つのでAは1になる。逆にAが1なら成り立たずAは0になるのだ。

●引き算を使った場合

```
A=1-A
```

IF文の場合と比べて、とても短くなった。Aが0だったら1に、0ならば1にちゃんとなる。このくらい短いと手軽に使えるテクニックといえる。

※スーパービギナーズ講座では今後の参考のため、ご意見、ご感想、ご質問などを募集します。また、初心者の方へのメッセージなども受け付けます。応募は必ずハガキで、MSX・FAN編集部「SB講座」係まで（住所は78ページ）。この記事で使用したハガキの投稿者には、Mファン特製テレカをさしあげます。

演算子とは

ところで、変数と計算などとひとくちにいっているが、これらをすべて把握するというのはけっこう至難の業なのだ。

変数はきちんとした理解さえあれば、AもF／もD1$も、どれもたいした差もなく使いこなせるだろう。しかし、計算のほうはどうだろう。MSXには性質の違うさまざまな計算方法があり、それぞれひとつずつ理解していく必要がある。

■計算は演算子を使っておこなわれる

「4と7を足すと、いくつですか?」と聞かれたら、あたまのなかには「4+7=」なんていう計算式が浮かぶだろう(もっとも、こんなかんたんな計算なら答えのほうが先かもしれないが)。この足し算を成立させているのが「+」という記号で、これら計算式に使う記号のことを、まとめて「演算子」と呼んでいる。

■演算子の種類

これら演算子には、足し算や掛け算など、おもに計算をおこなう「算術演算子」と、AはBより大きいとか、AとBは等しいといった関係を示す「関係演算子」、さらに数値の性質を条件によって比べ、演算結果をだす「論理演算子」とがある。

各演算子にはそれぞれ性質が決まっていて、計算結果も当然のことながら、それぞれ異なってくる。また、演算子にはその性質上、必然的に決まっている優先順位があり、MSXで計算をおこなうときには、このことをきちんと把握していないと、希望どおりの計算結果が得られない、なんてことになる。

各演算子の優先順位が、その性質から決まっているなら、性質をしっかりのみこんでおけば、ちょっと考えるだけで優先順位もわかることになる。なにしろたくさんある演算子の性質を覚えるだけでもたいへんなのに、さらに優先順位も覚えていたら、計算のまえに投げ出したくなってしまうだろう。

■それぞれの性質と優先順位

算術演算子は算数の授業でならうような計算がおもな役割で、優先順位は授業で習ったものとおなじだ。種類としては四則演算とべき乗、それに商と余りの計算用に「¥」と「MOD」がある。¥とMODは掛け算、割り算のつぎに優先される。

関係演算子とは、2つの値を比較して、そのときの関係が成立するなら−1、成立しないときには0を計算結果とする演算のことをいい、計算結果を比べたりするのに使用する。

論理演算子とは、数値の性質により値をもとめる計算のことで、プログラムを組めるようになるかならないかに大きく影響する重要なものだ。

各演算子については下のカコミでも説明しているが、これからも演算子についての解説はしていくので、覚えきれないからといってあきらめないように。

演算子の種類とその性質

演算子には、算術演算子、関係演算子、論理演算子とがあり、それぞれまったく違う計算方法をしている。なかでもわかりにくさナンバー1なのが論理演算子だろう。なにしろ、算術とか関係といった、名前でだいたい想像がつくほかの演算子と違って、論理などといわれても、いったいどんな演算をするのか、そんなにかんたんには想像することができない。まあ、この論理演算子だけは難物かもしれないが、IF文のなかの条件式やグラフィック命令でのロジカルオペレーションなどと、いろいろなところで顔をのぞかせるものだから、ここらできちんと理解しておけば、あとが楽になるぞ。

■算術演算子

算術演算子のおもな役目は、数値の計算ということになる。

数値の計算とひとことにいっても、優先順位の違う計算がいくつかあり、算術演算子のなかでもいちばん優先されるのがべき乗「^」だ。そしてそのつぎに掛け算「*」や割り算「／」があり、さらに整数除算「¥」と整数剰余「MOD」がある。いちばん優先順位が低いのは足し算「+」と引き算「−」だ。

これら算術演算子のなかで、いちばん抵抗があるのは整数除算と整数剰余だろう。なにしろほかの演算子は、学校の授業で習うのと、役割も優先順位もおなじだが、これらの記号は授業ではみかけない。しかし、小学校で割り算を習ったころのことを思い出してほしい。割り算の計算結果は、商と余りで答えていた。じつは¥はそれとおなじ働きをしていて、「¥」で割り算の商を、「MOD」でその余りが求められるのだ。

メモ：演算の優先順位は、カッコで囲まれた部分が最優先となり、ついですべての関数、つぎに算術演算子となる。算術演算子のなかの優先順位は、①^②*、／③¥、MOD④+、−となっていて、おなじ優先順位のものが並んでいるときは、左側のほうにある演算が優先される。

■関係演算子

関係演算子は、計算よりも、むしろ比較といったものが役割になっている演算子で、なぜ比較が演算になるのかといわれると、MSXではそうなっているとしか答えようがない。しかしそのために、スティック入力からいきなり座標増分を計算できたりと、いろいろ便利なところもあるのだ。

関係演算子の種類には、等号「=」と不等号「>」、「<」があり、どれも優先順位は変わらない。また、関係演算子による演算結果ははっきりしていて、関係が成立するなら−1を値とし、成立しないなら0を値としている。

メモ：関係演算子の表現例

AとBは等しい	A=B
AはBより大きい	A>B
AとBは等しくない	A<>B

■論理演算子

論理演算子はどちらかというとIF文などで条件式として使われることが多いが、たとえIF文のなかの条件式でも、論理演算はおこなわれている。だからしっかり理解していないと、思わぬバグのもとになってしまうのだ。

論理演算子を優先順位の高い順に列記すると、NOT、AND、OR、XOR、IMP、EQVの6種類があるが、IMPとEQVはめったに使われないものだ。

論理演算の仕組みはどれもおなじようなもので、どれかひとつ理解できればしめたもの。しかし、論理演算のやっかいなところは、演算がビット単位でおこなわれているところなので、数値を2進数として扱うことができなければ、論理演算の仕組みを理解するのがとても困難になってくる。

数値の2進数化というテーマは、とてもここでは教えきれないので、次回の講座のテーマとする。

メモ：論理演算子はIF文に使う条件式として用いられ、NOTは「〜でなければ」という意味になり、ANDは「〜と〜がともに成り立つ」、ORは「〜か〜が成り立つ」、XORは「〜と〜とのどちらか一方だけが成り立てば」という意味に置き換えることができる。

変数と演算子

演算子には算術演算子、関係演算子、論理演算子があり、それぞれ優先順位があった。

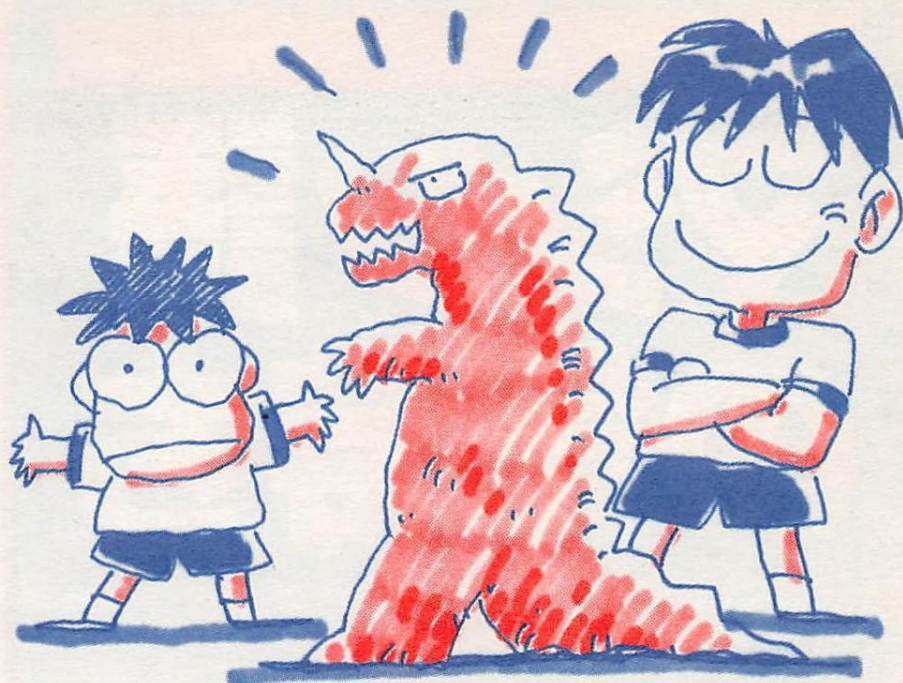
ここでは優先順位とひとくちにいっているが、これはよほど算数が苦手でなければ覚えるほどのものではない。なぜなら算術演算による計算があり、その結果を関係演算で比較し、論理演算で条件を判定するという、手順に基づいたものだからだ。

算術演算子と関係演算子は、小学校で習うていどの算数の知識があれば、あえて覚える必要はない。しかし、論理演算子のほうは、2進数に関する知識が必要になるので、紹介ていどの内容だったが、次回はもう少し実例をもとに解説するつもりだ。

まあ、演算子はこれくらいにしておいて、こんどは変数について解説しよう。

■変数の種類

すべての変数には、これは文字変数ですとか、整数型の変数ですといった「型」というものが存在し、その型によって扱えるデータが決まってくるのだ。変数の型には整数型「%」、単精度実数型「/」、倍精度実数型「#」、文字型「$」の4種類があり、MSXの電源を入れたばかりの状態では、すべて倍精度実数型になっている。また%や$などは型宣言文字といい、その文字のついた変数の型をあらわしている。このうち文字型はその名のとおり、文字を扱うことのできる変数のことだ。

文字型以外の変数は数値変数と呼ばれ、3つともおなじように数を扱うくせに、それぞれ性質が違う。そのなかで、整数型というのは−32768から32767までの整数を扱う変数のことで、小数を代入しても小数点以下は切り捨てられるし、この範囲を超える値にしようとすると、エラーがおこってしまうのだ。単精度や倍精度といった実数型の変数は、扱える数値の範囲が大きく、小数も使えるが、単精度で6桁、倍精度で14桁までが有効桁数となっている。それぞれ、有効桁数を超える桁があるときは、その桁で四捨五入されて有効桁数内におさめられてしまう。プログラムを打ちこむとき、「A=50000」など整数型の範囲を超える数値を入力すると、自動的に単精度型または倍精度型の実数を意味する型宣言文字がついてくる。

誤解のないように断っておくが、A%、A/、A#、A$といった、変数名がおなじで型が違う変数は、すべてぜんぜん別の変数なのだということ。だから、A%が1だからといって、A#が1とは限らないのだ。

変数と演算子との注意点

ここでは変数と演算子の組み合わせによって、エラーや誤動作が起きる場合を紹介しよう。

下の2つの図を見てほしい。図の上のほうは、組み合わせによりエラーや誤動作になる場合の例だ。

図の「TAND3」を例にとると、変数Tと3でANDをとるつもりが、TAN関数と変数D3というふうに、MSXにカン違いされてしまい、エラーや誤動作が起こるのだ。このように、MSXは左から順につづりを調べていき、予約語を見つけるとそこで確定させてしまう。だからそうならないように、変数と演算子の間に空白を入れたり、まぎらわしい演算をしないようにするくふうが必要になってくる。前例の「TAND3」なら「T AND3」にするか、変数Tの値をほかの変数に代入して演算すればいいのだ。

図の下のほうは、エラーや誤動作にならない場合の例で、たとえば「NOTO」は変数NOと予約語のTOとはカン違いされない。なぜならNOTのほうが先に予約語として判断されるからだ。

●エラーになる、または違う結果の出る場合

TAND3	XXOR3	DIMP2
↓	↓	↓
TAND 3	XXOR3	DIMP2

●正常な動作になり、間違われない場合

NOTO	2XORND(1)	D IMP2
↓	↓	↓
NOTO	2XORND(1)	D IMP2

```
PRINT 10 AND 3;10 OR 3
 2  11
Ok
```

↑「10 AND 3」と「10 OR 3」の論理演算をおこなったところ。答えは見てのとおり。片方の数値10を変数に代入して演算してみよう

```
T=10:PRINT TAND3
Syntax error
Ok
```

↑変数Tに10を代入して、変数Tと3のANDをとったところ。MSXは「TAND」の部分をTAN関数と誤解してしまったようで、エラーが出た

```
X=10:PRINT XOR3
Syntax error
Ok
```

↑こんどは変数Xに10を代入して、変数Xと3でORをしたら、こんどは論理演算子のXORと誤解されてしまい、またもやエラーが出た

```
XX=10:PRINT XXOR3
 3
Ok
```

↑変数XXに10を代入してOR演算をしたら、エラーは出ずに3と表示された。でも本当なら11になるはずで、X(内容は0)と3のXOR演算をした答えだ

スーパー
Super Beginners' 講座
ビギナーズ

超初心者

第10回

論理演算という大きな山を克服するためには、2進数という小高い丘を越えなければいけない。今回は2進数と論理演算の関係についてのお話だ。

2進数とは何か

世の中にはいろいろな数があるもので、MSXに用意された数をとってみても、ふだんみんなが使っている10進数をはじめ、2進数、8進数、16進数の4種類もの数がある。しかし、基本的には、2進数が理解できれば、8進数と16進数もかんたんに理解できるだろう。

それよりも重要なことは、MSXは2進数で数を数えているということだ。そのために、必ずといっていいほど2進数にはお目にかかれるはずだ。そんなときに、「なんだこりゃあ」とならないように、いまのうちにしっかり理解しておこう。

さて、2進数はどんなことに使われる数なのだろう。2進数が使われる典型的な例として、スプライトや文字のパターンデータを作るときがある。点がある、点がないという情報が、そのまま2進数の数値データとなっているのだが、知らず知らずのうちに使っていた人がいるんじゃないかな。

ところでMSXには、2進数がもっと手軽に使えるように、「BIN$」と「&B」というものが用意されている(右のリストと写真参照)。

BIN$は数値を2進数の形をした文字に変えてくれるもので、写真の例でいえば、10進数の「6」は2進数では「110」になることがわかる。

また&Bというのは、2進数を使うときに付けなければいけない記号で、これを付けないとたとえば「110」はそのまま10進数の110になってしまう。

!!キーボードにふれてみよう①

```
PRINT BIN$(6)
```

!!キーボードにふれてみよう②

```
PRINT &B10110001
```

```
PRINT &B10110001
 177
Ok
```

2進数の表現

MSXで2進数を扱うときは、その2進数のあたまの部分に「&B」というのを付けるのだ。これは、ふだん使っている数字と2進数とを区別するために決められた約束ごとなのだ。

&B1111111111111010

この16桁目の部分はサインビットと呼ばれていて、ここが1だと負(マイナス)の数、0だと正(プラス)の数という意味に使用されている。

この部分が2進数の数値を表す部分で、0と1の16桁で構成されている。それぞれの桁はビットとも呼ばれることがあり、1桁目がビット0、2桁目がビット1、……16桁目がビット15となっている。また、8ビット(8桁ぶん)で1バイトとも呼ばれる。

ふだんみんなが使っている数は10進数というもので、1つの桁では0〜9までの数字が使える。9を超えて10になると桁が1つ繰り上がる。2進数では、1つの桁で使える数字は0と1だけなので、1を超えるとすぐ桁が繰り上がる。だからちょっと大きめの数値を扱うと、すぐに何桁もの数になってしまうのだ。つまり、2進数という数は、計算したりするのには、あまり向いていない数だとわかるだろう。

●2進数の利点

16桁もある2進数でも、扱える数値の範囲はせまい。そんな不便な数が、なぜMSXに用意されているのかというと、ふだんみんなが10進数を使っているように、MSX(とは限らないが)では2進数を使っているからなんだ。こんな不便な2進数でも、使い方しだいでいろいろと便利なところもある。たとえば、パターンデータを作るときがいい例になるだろう。点(ドット)があるかないかの情報が、そのまま2進数の数値として使えるからだ。また、関係演算や論理演算も2進数と深く関わりがあるのだ。

※スーパービギナーズ講座では今後の参考のため、ご意見、ご感想、ご質問などを募集します。また、初心者の方へのメッセージなども受け付けます。応募は必ずハガキで、MSX・FAN編集部「SB講座」係まで(住所は79ページ)。この記事で使用したハガキの投稿者には、Mファン特製テレカをさしあげます。

2進数と論理演算

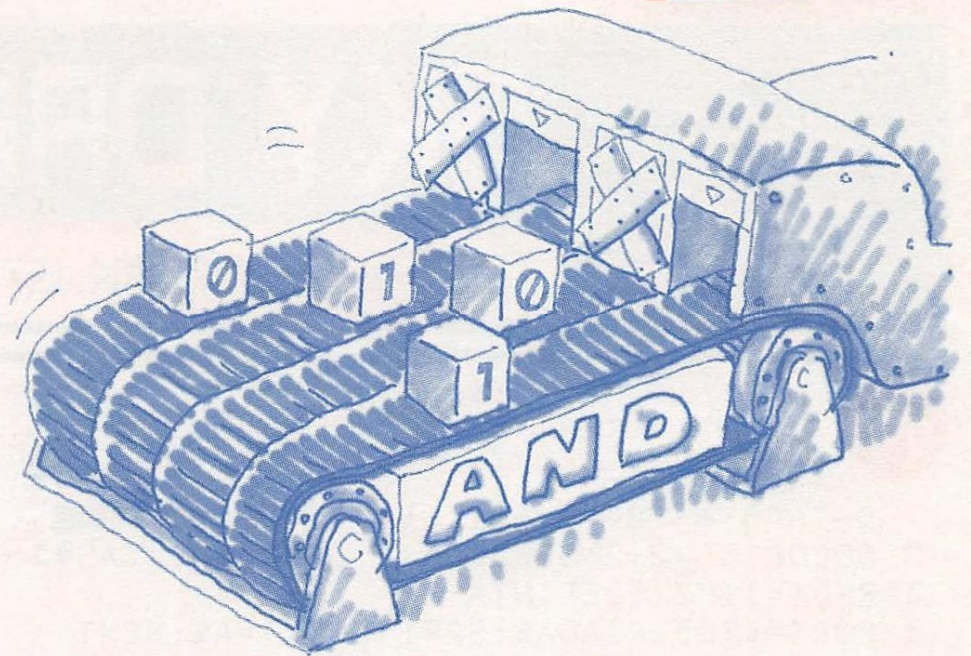
2進数で扱える数値は、32767から−32768までの整数に限られている。16桁目(ビット15)が1になっていると、負の数値として扱われる。このとき、0からいくつ引かれたものかで負の数の大きさを表している。だから、「&B1111111111111111」なんて数をみると、ついいちばん大きいように思えるけど、じつは−1なのだ。なぜなら、この16桁すべてが1のときに1をたすと、すべての桁で繰り上がりをして、結果は全部Øになるので、−1になるというわけだ。

■2進数と式の関係

−1といえば、関係式や条件式が成り立ったときに、この値になるのを知っているかな? じつはこれらの式は、2進数と深い関係があるのだ。

条件式に使われている論理演算子には、NOT、AND、OR、XOR、IMP、EQVの6種類があり、それぞれ意味がある。ただ、共通していえることは、これらの演算は対象となる数値を2進数とみなして行うことと、各桁(ビット)ごとに計算していることだ。

各桁ごとの計算というのは、繰り上がりや繰り下がりのない計算という意味だ。各論理演算子による計算内容は、下にまとめてあるので読んでほしい。

ところで、関係式や条件式が成り立つと、どうして−1になるかわかったかな? 答えはとてもかんたんで、ここでの判定結果は、論理演算とおなじように2進数で出しているからなんだ。式が成り立つときは、すべてのビットを1にしているので

結果は−1になり、式が成り立たないときは、すべてのビットがØになるので結果は0になるという仕組みなのだ。

ちょっと余談になるが、関係式や条件式の判定結果が、数値として得られるということは、計算式のなかで利用できるという利点がある。使い方しだいだが便利なことが多いのだ。たとえばスティック入力から計算式だけで座標にできたり、ハイスコアが計算式で更新できたりなど、おかげでいろんなことが計算式だけでできるからなのだ。

判定結果が数値で得られる利点を大いに活用するためには、きちんと2進数と論理演算の関係を理解しておく必要があるので、次回の講座では、ちょっとしたプログラムを使ってその関係を体験してみよう。

論理演算による各ビットの変化

■NOT X

Xのビットn	1	Ø
演算による変化	Ø	1

NOTは条件式では否定となる。この演算は、左の図のように、与えられた数値を2進数として、それぞれの桁(ビット)ごとに反転していく。Øならば1に、1ならばØにしていくのだ。与えられた数値と演算結果を足すと、すべてのビットが1になるので−1になる。

■X AND Y

Xのビットn	1	1	Ø	Ø
Yのビットn	1	Ø	1	Ø
演算による変化	1	Ø	Ø	Ø

ANDは条件式では、両方とも成り立つかの判定に使用される。演算では、両方の数値の各ビットを見比べて、どちらも1のときだけ結果のビットも1になり、それ以外のときはビットはØになる。これもよく使うので、ORと組みにして覚えるといいだろう。

■X OR Y

Xのビットn	1	1	Ø	Ø
Yのビットn	1	Ø	1	Ø
演算による変化	1	1	1	Ø

ORは条件式では「または」という意味になる。この演算は、2つの数値の各ビットごとに、どちらかが1になっていれば1、両方ともØのときだけØに、演算結果のビットを作っている。よく使うので、「ORはØ−Ø以外は1になる」と覚えておけばいいだろう。

■X XOR Y

Xのビットn	1	1	Ø	Ø
Yのビットn	1	Ø	1	Ø
演算による変化	Ø	1	1	Ø

XORは片方だけ成り立つときの判定で使用する。演算では、両方の数値の各ビットを見比べていき、おなじであればØ、そうでなければ1に、演算結果のビットを決めていく。演算として使用されることが多く、「XORはおなじならØ」と覚えておくといい。

■X IMP Y

Xのビットn	1	1	Ø	Ø
Yのビットn	1	Ø	1	Ø
演算による変化	1	Ø	1	1

IMPは条件式としても、演算としても、あまり使われることがない演算子だ。これは、この演算子の利用法がよくわからないということと、ほかの論理演算子でいくらでも代用できるという理由からだろう。覚えたい人は「1−Øのときだけ Ø」と暗記するといい。

■X EQV Y

Xのビットn	1	1	Ø	Ø
Yのビットn	1	Ø	1	Ø
演算による変化	1	Ø	Ø	1

EQVは「どちらも」という意味になるが、これははっきりいってすぐ存在を忘れるくらい使わない。XORの反対の動作をするだけなので、覚えなくても不都合なことはまるでないだろう。覚えたいなら、「EQVはおなじなら1、ちがうならØ」とでも覚えよう。

スーパー ビギナーズ
超初心者 Super Beginners' 講座 第11回

今月は論理演算の抜き打ちテストをやる。なんてのはうそで、遊び感覚で論理演算のしくみに触れられる、かんたんなゲームを作ったのでさっそく遊んでみてほしい。

論理演算のしくみを知ろう

論理演算のだいたいのしくみや理論的な部分はわかったつもりでいる、でも、いまいちピンとこない……というのが正直な実感ではないだろうか。

たとえば、1のNOTがØでØのNOTが1、のはずなのに
PRINT NOT Ø
を実行すると、－1になるし、
PRINT NOT 1
なんかは－2になる。なぜ?

じつは、1のNOTがØ……というのは、ビット(2進数1桁)単位の話で、じっさいの論理演算は、整数型数値の枠(16ビット)いっぱいを使っておこなわれているのだ。だから、たんに「0」といっても16個のØのならぶ2進数、「1」も15個のØと1個の1でできた2進数だと考えなくてはいけない。そして1桁ごとに「NOT」をしていくと……。わかったかな?

わかった人も、まだピンと来ない人も、今月の付録ディスクに入っている「SB－KOZA.BAS」というファイルを実行してみてほしい(遊び方は下のカコミ。ディスクユーザーでない人がいたら、ごめんなさい。リストは掲載していません)。

テストのような、ゲームのような、このプログラムは、論理演算のシミュレーションなのだ。

キーボードにふれてみよう

RUN"SB-KOZA.BAS

ディスクを抜いてMSX本体の電源を入れ(ターボRは起動するまで1キーを押しておく)BASICの初期画面が出たら付録ディスクを入れ、上記の命令を打ちこんでリターンキーを押そう。ゲームがはじまる。

```
SCORE    0   HI-SCORE    0
ﾀﾞｲ 1ﾓﾝ
   19272 OR      -79 ?
         HEX   BIN
 19272   4B48  0100101101001000
   -79   FFB1  1111111110110001
  -128   FF80  1111111110000000

                    TIME= 86
```

付録ディスクのプログラムについて

付録ディスクのなかの「SB－KOZA.BAS」を実行すると『論理演算ゲーム』がはじまる。

■基本的な遊び方

まず、緑色の枠のいちばん大きなスペースに、論理演算式の問題が表示される。画面右下でタイマーが動きはじめるので、300になるまでに解答(後述)する。タイマーが300になるまえに解答できたときはリターンキーを押す。

正解すると得点が表示され(ヒントの有無で判定)次の問題に進むときはカーソルキーの下を押す。

全20問の出題を終わると、総合評価(正解数/ヒントを見た回数/総合得点)を表示。Yキーで再挑戦。Nキーで「おつかれさま」。

■解答のしかた

解答は、すべて16桁の2進数の各ビットを切り換えることでおこなう。緑枠の右下の欄、16個の0のならんだところが解答欄だ。

その上に、2列(NOTのときは1列)の16桁2進数があるが、これは問題に使われている数値を2進数化したもの。これを見て、各桁で指定された論理演算をしていくのだ。

明るい青のカーソルをカーソルキーで左右に移動させ、スペースキーを押すと、そのときカーソルの指している桁の1、0が切り換わるしくみだ。

ESCキーを押すと、そのときに指定されている論理演算のビット変化のパターンがヒントとして表示されるので、各演算の意味を忘れていてもここで記憶を新たにできる親切設計なのだ。

```
SCORE    5   HI-SCORE    5
ﾀﾞｲ 2ﾓﾝ
  -32040 IMP -13620 ?
         HEX   BIN
-32040   82D8  1000001011011000
-13620   CACC  1100101011001100
-13620   CACC  1100101011001100
  ﾋﾝﾄ:IMP
      < ｶｸ ﾋﾞｯﾄ ﾉ ﾍﾝｶ >
      0 IMP 0 ===> 1
      0 IMP 1 ===> 1
      1 IMP 0 ===> 0
      1 IMP 1 ===> 1
                   TIME=105
```

【画面の見方】緑枠のいちばん大きなスペースが出題欄。その下に出題に使われた数値と解答の数値を、左から10進数、16進数(HEX)、2進数(BIN)で表示している。ESCキーを押すか、失敗するかすると、画面下にヒント(各ビットの変化)が表示される。

超初心者 Super Beginners' 講座（スーパービギナーズ講座）

第12回

今月も10月号に続いて論理演算のおはなしだ。ただし、計算式や条件式で使われている論理演算子とはちょっとちがい、グラフィックに関係した論理演算を紹介する。

MSXFAN 付録ディスク →P116の「スーパー付録ディスクの使い方」参照

2つのLINE文で5つの四角形

計算式などで使う論理演算はいちおう10月号で終わりにし、今月はちょっとちがった目的に使われている論理演算を題材にしたいと思う。ただし、題材となる論理演算はSCREEN5以上の画面でしか使えないので、MSX1のユーザーの人たちは、「MSX2以上ではこんなことができるのか」ていどに読んでおいてほしい。

■2つのLINE文で、5つの四角形を作る

マッチ棒パズルなんかよりずっとかんたんなこの問題。MSXの画面にLINE命令を使って、見た目には5つ四角があるように2つの四角形を表示してほしいというものだ。

小学校の低学年の人でもわかるかもしれないくらい、かんたんな問題なんだけど、LINE命令を使って四角を表示する方法を知らない人もいるだろう。そういう人のためにかんたんに説明しておこう。

LINE命令は、指定された色で直線を引く命令で、最後にBを付けると長方形を表示し、BFなら中が塗り潰された長方形を表示する命令だ。LINE命令の書式についてはマニュアルに必ず載っているので、そこを見てほしい。

さて解答はというと、右の写真のとおり。そしてその画面を作っているのが、すぐ上にあるリスト①だ。最後にあるFOR～NEXTループは表示したあとテキスト画面に切り換わるのを防ぐために、たんに入力待ちの状態にしているだけ。

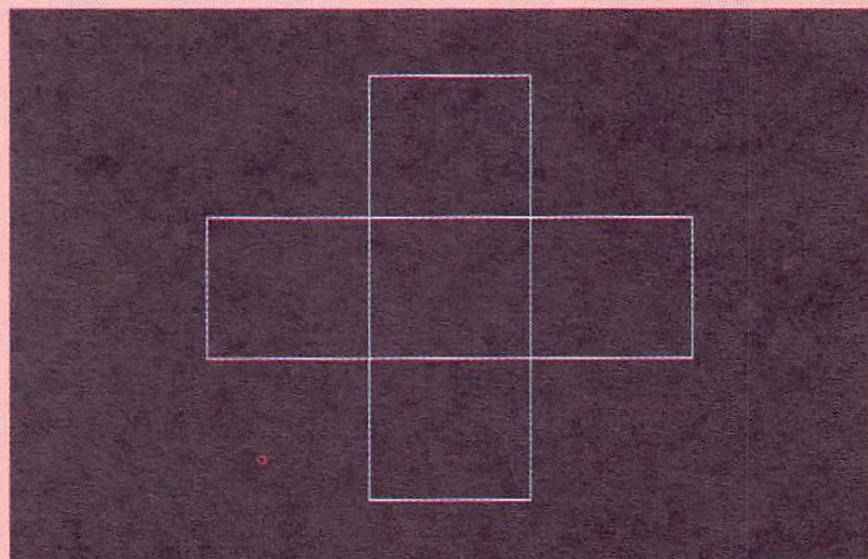
では次に、おなじように2つのLINE文を使って5つの四角を作ってほしい。ただし、今度は線ではなく、色で塗られた四角を使って解答せよ。PAINT命令を使ったら反則だぞ。

右下の写真はその解答例だ。すぐ上のリスト②を見ると、さっきのリストとあまり変わらないのがわかる。変わっているのは、表示色とLINE命令の最後にあった「B」が「BF」になっていることくらい。おや？　2つめのLINE命令の最後に「OR」というのが付いているぞ。

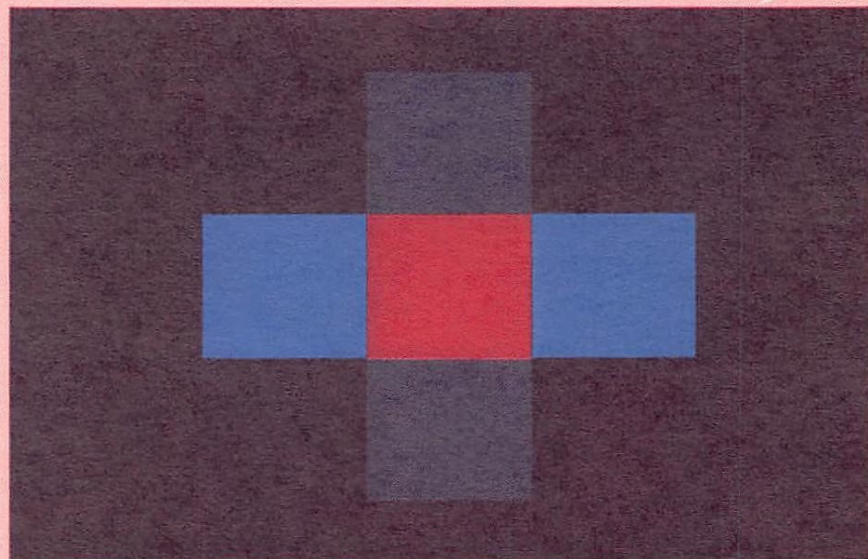
ORといえば、論理演算子のなかにもおなじものがあったのを覚えているかな？　このORこそ目的のグラフィック用の論理演算のORなのだ。

リストにある「，OR」の部分を空白にして実行してみよう。ORがないと、短冊に切った折り紙を十字型にかさねたような画面になるだろう。しかし、ORがあれば折り紙に色セロファンを重ねたみたいに、下の色が影響して実際に表示される色が変わるのだ。ORの働きがひとめでわかるだろう。

キーボードにふれてみよう①

```
COLOR15,0,0:SCREEN5:LINE(50,70)-(200,120
),15,B:LINE(100,20)-(150,170),15,B:FORI=
0TO1:I=-STRIG(0):NEXT
```

↑これが「2つの四角で5つの四角を作る」の解答例だ。ここでは見栄えのいいように形を作っているが、もちろんたて長でもよこ長でも、四角が5つあればOKだ

キーボードにふれてみよう②

```
COLOR15,0,0:SCREEN5:LINE(50,70)-(200,120
),2,BF:LINE(100,20)-(150,170),4,BF,OR:FO
RI=0TO1:I=-STRIG(0):NEXT
```

↑「2つの塗りつぶしの四角を使って5つの四角を作る」の解答例。論理演算子はORでなくてもかまわないが、色の組み合わせをよく考えないと失敗することもある

グラフィックに使える論理演算

グラフィック関係の論理演算子には、ORのほかにPSET、PRESET、AND、XOR、TPSET、TPRESET、TAND、TOR、TXORの合計10種類がある。計算式で使っていた論理演算子より種類が多いのは確かだが、実際にはこっちのほうが覚えやすいのでそんなに心配しなくていい。

■各論理演算子の性質

全部で10種類あるが、後半の5種類は前半のものに「T」を付けただけだと気付いたかな? Tのあるものとないものでは基本的に性質は変わらないが、Tが付いていると、指定された色(カラーコード)が0の透明色だった場合は、もともとそこにあった色が表示されるようになっているというものだ。

■カラーコード表

COLORコード		初期パレット成分値		
10進	2進	赤	緑	青
0	0000	0	0	0
1	0001	0	0	0
2	0010	1	6	1
3	0011	3	7	3
4	0100	1	1	7
5	0101	2	3	7
6	0110	5	1	1
7	0111	2	6	7
8	1000	7	1	1
9	1001	7	3	3
10	1010	6	6	1
11	1011	6	6	4
12	1100	1	4	1
13	1101	6	2	5
14	1110	5	5	5
15	1111	7	7	7

では各論理演算子の性質をかんたんに説明しよう。

●**PSET** 指定された色をそのまま表示する。論理演算子を省略した場合も同様になる。

●**PRESET** 指定色のNOTとなる色が実際に表示される。

●**AND** 指定色と、もとの色とでAND演算をし、その結果が表示色になる。

●**OR** 指定色と、もとの色とでOR演算をし、その結果が表示色になる。

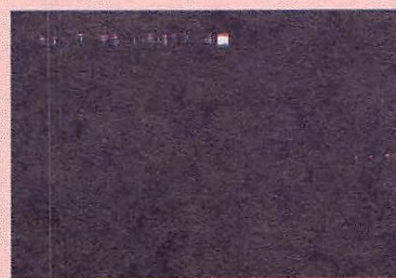
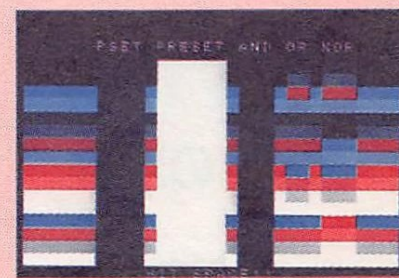
●**XOR** 指定色と、もとの色とでXOR演算をし、その結果が表示色になる。

グラフィックでの論理演算は、演算対象がカラーコードだという点をのぞけば計算式などに使われる性質とほとんど変わりがない。ただし、演算対象がカラーコードなので、0〜15までの4ビット(SCREEN8では8ビット)が演算対象となり、計算式のように0のNOTは−1ではなく、ここでは15(または255)となるので注意してほしい。

色がどのように変わるかピンとこない人は、右上のリスト③を実行してみてほしい。付録ディスクには、ファイル名SAMPLE−3. SBZで入っているので実行してみるといいかも。

あまりたいしたものではないが、最初の入力で指定する色コード(0〜15)を入力すると、画面に上から順に色のオビが表示され、その色が指定色と論理演算によって変化するのを目で確認できるというものだ。

!!キーボードにふれてみよう③

```
10 COLOR15,0,0:DEFINTA-Z:OPEN"grp:"AS#1
20 IFINKEY$<>""THEN20ELSESCREEN1:WIDTH29
:INPUT"ﾁｪｯｸ する いろは?";A:IFA<0ORA>15THEN20
30 SCREEN5:FORI=0TO15:LINE(0,I*10+30)-(2
55,I*10+39),I,BF:NEXT:PSET(50,16),0,PSET
:PRINT#1,"PSET PRESET AND OR XOR"
40 LINE(50,30)-(81,189),A,BF,PSET
50 LINE(90,30)-(137,189),A,BF,PRESET
60 LINE(146,30)-(169,189),A,BF,AND
70 LINE(178,30)-(193,189),A,BF,OR
80 LINE(202,30)-(225,189),A,BF,XOR
90 PSET(84,192),0,PSET:PRINT#1,"HIT SPAC
E!!":FORI=0TO1:I=-STRIG(0):NEXT:GOTO20
```

↑変化を調べたい色のカラーコードを入力してリターンキーを押す

↑いちばん左がもとの色で、演算名の下は指定した色での変化になる

!!キーボードにふれてみよう④

```
C=15:R=7:G=6:B=5:COLOR=(C,R,G,B)
```

■カラーコードに登録された色を変える(パレット切り換え)

ところで、カラーコード1はつねに黒、15は白と決まっていると思っている人がいるんじゃないかな? じつは、MSX2以上の機種では、カラーコード1〜15に登録されている色の情報を、自分で自由に変えられる。④を実行してみよう。白の色が変わるぞ。Cにはカラーコード、R、G、Bにはそれぞれ赤、緑、青の成分(0〜7の範囲)が入るのだ。ついでに、左にある表は初期状態のパレット成分値を記したものだ。どの成分を多くすると何色に見えるかの参考にするといい。また、パレットを初期値にもどすには、SCREEN文を実行して画面モードを変えるか、COLOR=NEWまたは、COLORのみを実行すればOKだ。　(MORO)

論理演算の使えるグラフィック命令

グラフィック画面で論理演算子を使うことのできる命令には、今回扱ったLINE命令のほかに、以下の4つがある。

●PSET　指定された色で画面に点を表示する命令。色を指定しなければ、そのときの前景色で点を表示する。

●PRESET　指定された色で画面の点を消す命令。色を指定しなければ、そのときの背景色で点を消す。

●PUTKANJI　指定された色で漢字を表示する命令。ただし、漢ROMがないと表示されない。

●COPY　あるグラフィックを画面のほかの部分に複写したりできる命令。論理演算とこの命令を組み合わせるとかなり使えるのだ。次回のテーマ候補の命令。

!!キーボードにふれてみよう⑤

```
10 COLOR15,0,0:SCREEN5:OPEN"GRP:"AS#1
20 FORI=0TO1:PRESET(100+I,100)      :PRINT
#1,"ABC":NEXT
30 GOTO30
```

左の写真は文字を1ドットずらして2回表示し、太文字のように見せようとしたところ、失敗した例。点のない部分も画面に書きこまれたのが原因だ。

!!キーボードにふれてみよう⑥

```
10 COLOR15,0,0:SCREEN5:OPEN"GRP:"AS#1
20 FORI=0TO1:PRESET(100+I,100),,OR:PRINT
#1,"ABC":NEXT
30 GOTO30
```

左の写真はSCREEN5以上のグラフィック画面での太文字表現に成功した例で、リストのPRESET命令の最後にある「OR」がポイントだ。

スーパー
Super ビギナーズ Beginners' 講座

超初心者

第13回
グラフィック入門その1

MSX FAN 付録ディスク
→P116の「スーパー付録ディスクの使い方」参照

今回からSCREEN5以降のグラフィック画面について紹介していきたいと思う。その第1回として、グラフィック画面とはどんなものかを紹介しよう。

MSX2の画面モードの種類

MSX1からMSX2になり画面表示が強化されたのは、もうだいぶ前のことだ。しかし、せっかく画面表示が強化されていても、実際にはあまり使いこなされていないようだ。これはMSX本体に付いているマニュアルでは、十分に使えるだけの知識が得られないことも原因ではあると思うが、いちばんの理由は、どんなことができるのか知らない人が多いためではないだろうか。もしこれが的中しているなら、知らなかった人はとても不幸だと思うので、これからしばらくのあいだは、この強化されたグラフィックとスプライトを扱っていこうと思う。でも今回は、グラフィック画面を扱うための基礎知識とでもいうか、グラフィック画面とはどんなものかを紹介しよう。

■画面モードの種類

MSXの画面モードには、大別してテキスト画面とグラフィック画面の2種類がある。

テキスト画面とは文字を表示するのに都合のよいような作りになっていて、絵を描いたりするのは不得意な画面のことで、ふだんみんながプログラムを打ちこんだりしているとき、打ちこんだ文字が表示されている画面がテキスト画面なのだ。

小さいころ、磁石つきでプラスチック製の数字やひらがなを、白いボードに張り付けてよろこんでいた経験のある人は多いと思うが、テキスト画面とはちょうどそんな感じになっていて、数字とかひらがなとか、形の決まっているものを画面に置いていっているようなものなのだ。どんなにカンの鈍い人でも、とてもこれでは絵は描けないとわかるだろう。少なくとも、上手な絵は期待できないのだ。

グラフィック画面はその逆で、絵を描いたりするのにはとても都合がいいようになっている。グラフィック画面を例えれば、無地の画用紙に色えんぴつを使って絵を描くようなもので、ウデさえあれば、素晴らしい名画だって夢ではない(?)だろう。もちろん文字の表示もできるのだけど、ちょっと面倒な手続きが必要になる。テキスト画面のように、ポンと文字を置くというようにはいかないのだ。

■2つのスプライトモード

じつはグラフィック画面でも、使えるスプライトモードによって2種類にわかれている。

SCREEN2と3は、MSX1とも互換性のあるスプライトモード1が使われているが、SCREEN4以降ではスプライトモード2というのが使われているのだ。

今月の付録ディスク「ファンダム・サンプルプログラム」にスプライトモード2を使ったかんたんなデモを入れておいた。動いている3つの円は、それぞれ1枚のスプライトなのだ。メニューから実行できるので、スプライトモード2がどんなものか見ておいてほしい(ファイル名：SB-KOUZA．SB1)。

●テキスト画面
SCREEN0と1がテキスト画面と呼ばれるもの。SCREEN0は真面目にテキスト画面をしていて、ほとんどプログラムを打ちこむ以外には使わないだろう。SCREEN1のほうはやや不真面目で、スプライトの表示や8文字ごとの色設定などができる。

●グラフィック画面
SCREEN2以降の画面はグラフィック画面と呼ばれるもので、おもに絵を描いたりするのに使用する。SCREEN命令でグラフィック画面に変えるのだが、プログラムが終了するとすぐにテキスト画面にもどるので、そうならないためのくふうが必要になる。

●画面モードとスプライトモード
スプライトモードが1になるか2になるかは、画面モードによって自動的に決まる。また、おなじグラフィック画面でも、表示ドット数や使える色の数などが決まっているので、自分がやろうとしていることにあわせて画面を選ばなくてはいけないのだ。

グラフィックモードの利点

画面モードにはテキスト画面とグラフィック画面があり、グラフィック画面でもスプライトモードによって2つにわかれていた。つまり、MSXにはおおざっぱにわけて3種類の画面モードがあるということだ。ただし、これはMSX2でのおはなしで、MSX2+以降では自然画モードと漢字モードが加わって、さらにややこしくなる。

ここではSCREEN5〜8の画面について、その魅力についてふれてみよう。

■グラフィックモードの利点

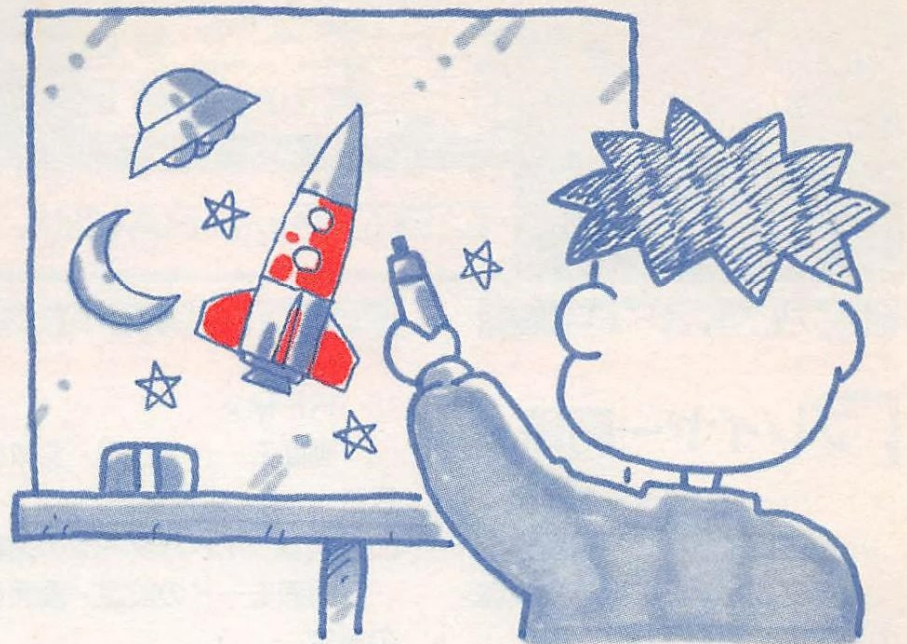
SCREEN5以降の画面では、グラフィック操作の命令が充実していて、COPY命令やSETPAGE命令、境界色を指定できるPAINT命令、そして、前回紹介したグラフィック命令で使える論理演算子などがあり、こういった命令の組み合わせでアニメーションやCGツールなどが、けっこうかんたんに作れてしまったりするのだ。

そして、パレット切り換えやハードスクロールなども、ほかの画面でも使えるが、やっぱりSCREEN5以降の画面で使うのがいいようだ。たあいもない落書きのような絵でも、これらを使えばいいデモになったりしてしまうから不思議だ。

そして、SCREEN5以降の画面では、自動的にスプライトモード2が使われる。しかし、スプライトモード2とはいったいどんなものなのだろうか？

スプライトモード2の機能をスプライトモード1と比べながら紹介しよう。

■スプライトモード2の利点

SCREEN1なんかでスプライトを表示するとき、横に5つ以上ならべて表示しようとしても、画面にはスプライトが4つまでしか表示されない。ところがスプライトモード2では、スプライトを横に8つまでならべて表示することができるのだ。このことを知っている人はけっこういるのだが、どうもスプライトモード2とは、ただそれだけと思っている人が多い。

つぎにあげるのは大きな特徴で、スプライトモード1ではスプライトを1色でしか表示できなかった。これに対して、スプライトモード2では、スプライトのドット横1列ごとに1色ずつ指定できるのだ。だから、下のキーボードにふれてみようのように、1枚のスプライトを虹色にすることもできる。

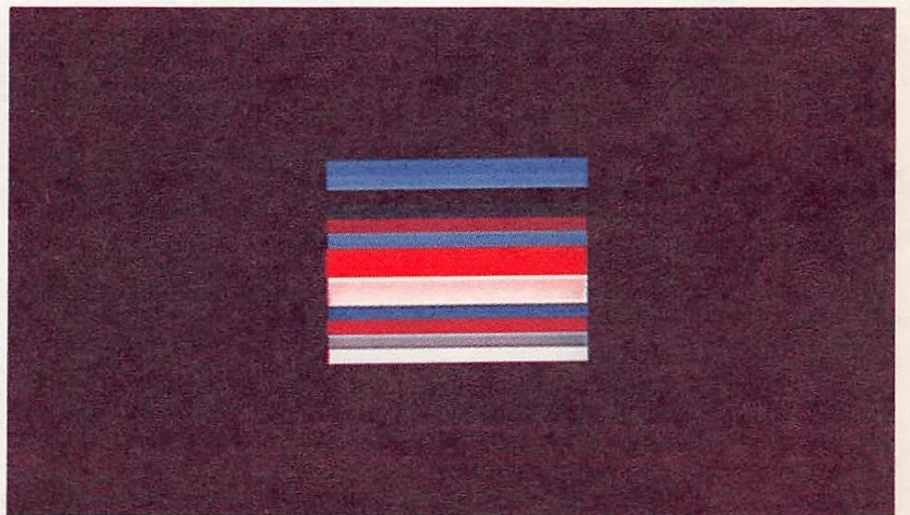
また、スプライトモード1では2枚のスプライトを重ねてもせいぜい衝突が起こるぐらいのものでしかなかったが、スプライトモード2ではスプライトが重なった部分の色をおたがいの色のORにすることができ、衝突判定をするかしないかも設定できるのだ。これもスプライトのドット横1列ごとに指定できるので、シューティングゲームの自機の胴体の部分だけ当たり判定をする、なんてこともできてしまうのだ。

スプライトの色と衝突判定の設定はCOLORSPRITE命令を使って設定する。詳しい設定の方法などは、次回のこの講座でやろうと思っている。

‼キーボードにふれてみよう

下のリストを打ちこんで、RUNすると、画面の中央付近に横じま模様のスプライトが表示される。リストを見るとわかるが、1つのスプライトで多色刷りをしているのだ。以下にこのリストの解説を掲載しよう。

10 画面モードとスプライトサイズ設定／画面の色設定／画面消去
20 スプライトパターン定義⇨ただの四角形に定義している
30 スプライトのカラーデータ作成⇨データについては次号で紹介する
40 スプライトの色定義
50 スプライト表示⇨色は定義してあるので、指定しない
60 トリガー入力待ち

```
10 SCREEN5,3:COLOR15,0,0:CLS
20 SPRITE$(0)=STRING$(32,255)
30 FORI=0TO15:A$=A$+CHR$(I):NEXT
40 COLORSPRITE$(0)=A$
50 PUTSPRITE0,(100,100),,0
60 IFSTRIG(0)=0THEN60
```

⇧COLORSPRITEを使って、多色刷りのスプライトを表示してみた。行40を消して実行してみれば、全然ちがうことがわかるだろう。次回はちゃんと形のあるものを見せるぞ

●MSX2+以降の画面モード

MSX2+以降の機種では、自然画モードと漢字モードがある。自然画モードはSCREEN8の色数にYJK方式というものをミックスしたグラフィック画面のことで、表現できる色数が大幅にアップしている。漢字モードのほうは見た目は漢字の使えるテキスト画面といったところなのだが、じつはちょっとイタズラすれば、絵も描けてしまうという変わり者だ。

●グラフィック命令

SCREEN5以降では、COPY命令、SETPAGE命令、境界色の指定できるPAINT命令などが使えるようになる。

COPY命令はグラフィックのある部分を、画面のどこかやRAM、ファイルに複写できる命令で、その逆も可能だ。また、論理演算子も使えるので重ねあわせなどができる。

SETPAGE命令、というよりもSCREEN5以降では、絵の描ける画面が複数存在している。だから、この命令でそれぞれのページを切り換えて、アニメーションや絵柄のパーツの保存場所などに活用できるのだ。

PAINT命令はSCREEN2〜4でも使用できたが、赤い円の中を赤く塗ることはできても、青く塗ることはできなかった。それがSCREEN5以降では可能になったわけだ。

スーパー
Super Beginners' 講座
ビギナーズ

超初心者

第14回
グラフィック入門その2

MSX・FAN 付録ディスク
→P116の「スーパー付録ディスクの使い方」参照

前回はMSX2以降で強化されたグラフィック関係の、とくにSCREEN5以降の画面でのさまざまな特典を紹介した。その特典のなかの1つであるスプライトモード2について、今回から扱っていこうと思う。

COLORデータの設定

前回は、SCREEN4以上の画面モードでは、スプライトを横1列ごとに色付けしたり、スプライト同志の衝突を無視させられるなど、MSX1にはなかった特典を紹介した。

今回はそれらを可能にするための、COLORSPRITEという命令を紹介しようと思う。

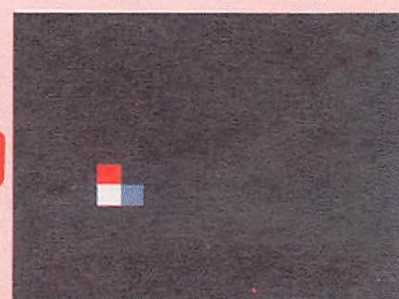
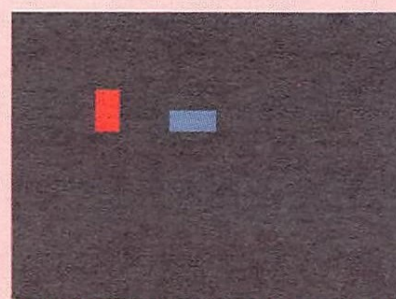
■データ設定には2種類ある

この命令で設定するデータは図1のように、下位4ビットがカラーコード、そして機能に関する設定となる部分が上位4ビットのうち3ビット(図のEC、CC、ICの部分。ここでは「機能ビット」と呼ぶ)となっており、各機能ビットは1のときに有効となる。

データの設定には図2のように2とおりの方法があり、上はスプライト面全体に対する命令で、下はスプライト面の横1列ごとの設定となる。データの設定で注意することは、スプライト面全体に対する設定は数値型データで行っているが、横1列ごとの設定では文字列データで行っていることだ。また、データが文字列のとき、COLORSPRITE命令に「$」が付くことも忘れないでほしい。

キーボードにふれてみよう

下のリストを実行すると、赤と水色の2枚のスプライトが表示される。ここでスペースキーを押すとスプライトが2枚重なって下に動きだす。行80のスプライトを動かしている部分では、赤いスプライトだけを動かしているのだが、行50で設定されるデータに64を足し、CCビットを1にして優先順位をなくしているので、2枚とも動いてしまうのだ。

```
10 COLOR15,1,1:SCREEN5,3
20 SPRITE$(0)=STRING$(16,255)
30 SPRITE$(1)=STRING$(8,0)+STRING$(8,255
)+STRING$(8,0)+STRING$(8,255)
40 COLORSPRITE(0)=8
50 COLORSPRITE(1)=7+64
60 FORI=0TO1:PUTSPRITE I,(50+50*I,50):NE
XT
70 BEEP:FORI=0TO1:I=-STRIG(0):NEXT
80 FORI=0TO211:PUTSPRITE 0,(50,(I+50)MOD
212):NEXT:GOTO80
```

↑先月掲載されたファンダムのプログラム「アシカの修業3」で使われていた方法だ

■図1　設定するデータの内容

EC	CC	IC	Ø	C1	C2	C3	C4

n列目のデータ

- 未使用
- 色コード(Ø〜15)
- ICビット………このビットが1のときはスプライトの衝突判定を無視するようになる。
- CCビット………ここが1だとスプライト面の優先順位がなくなり、衝突判定も無視する。詳しくは右ページの本文を参照。
- ECビット………画面の左端にスプライトを表示するとき、ここが1だと実際の表示位置が左に32ドット移動する。

スプライト面m
1列目
2列目
n列目

■図2　COLORSPRITEの書式

●スプライト面全体のデータ設定

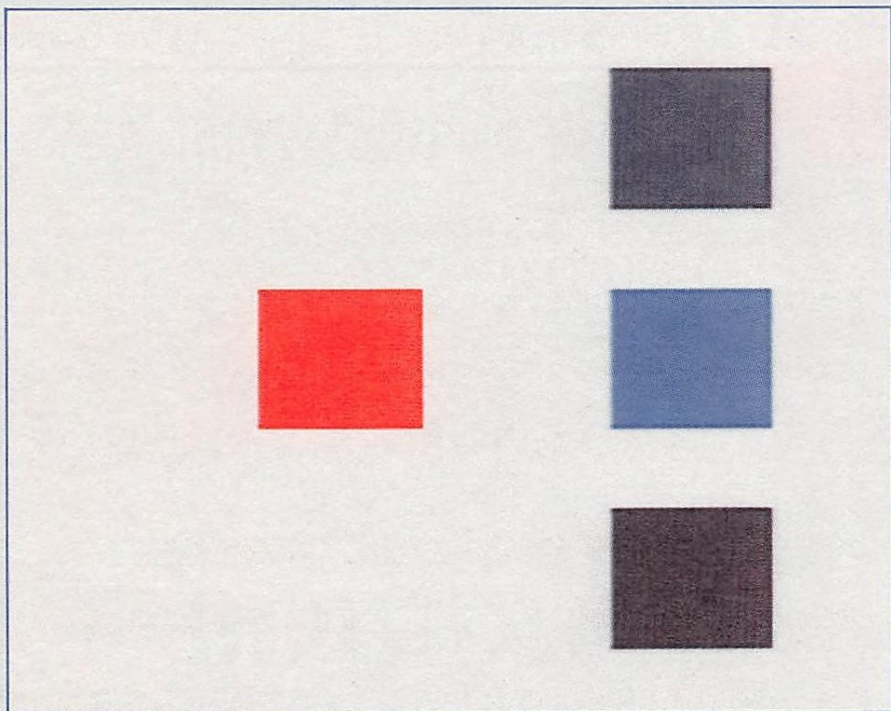
COLORSPRITE(〈面番号〉)=n

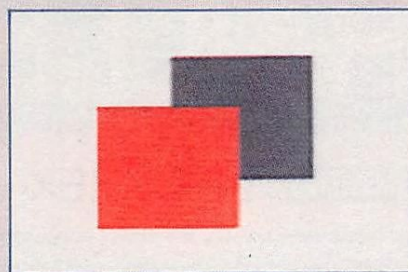
●横1列ごとのデータ設定

COLORSPRITE$(〈面番号〉)=CHR$(n1)+CHR$(n2)+……+CHR$(n16)

1列目のデータ　2列目のデータ　……………16列目のデータ

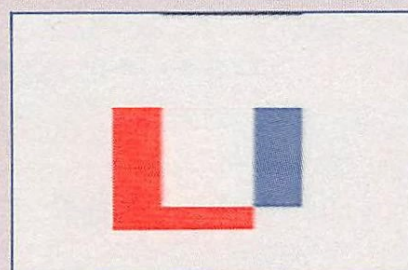
■スプライトの重なりによる機能ビットの働き

```
10 COLOR15,15,15:SCREEN5,3
20 SPRITE$(0)=STRING$(32,255)
30 COLORSPRITE(1)=32+1
40 COLORSPRITE(2)=64+2
50 FORI=1TO8:X(I)=(I>1)*(I<5)+(I>5):Y(I)
=(I<3)+(I>3)*(I<7)+(I>7):NEXT
60 PUTSPRITE 1,(140,140),,0:PUTSPRITE 2,
(140,90),,0:PUTSPRITE3,(140,40),4,0
70 ONSPRITEGOSUB100:X=70:Y=90
80 SPRITEOFF:S=STICK(0):X=(X+256+X(S))MO
D256:Y=(Y+212+Y(S))MOD212:SPRITEON:PUTSP
RITE0,(X,Y),8,0
90 IFSTRIG(0)=0THEN80ELSECOLOR,4,7:END
100 SPRITEOFF:BEEP:RETURN
```

3つの機能ビットのうち、CCビットとICビットはスプライト同志が重なったときにその効力が見られるものだ。そこで、上のようなサンプルリストを掲載してみた。これは、機能ビットを使っていないふつうのスプライトと、それぞれタイプの違う3つのスプライトとが衝突するとどうなるかを試せるものだ。まず、プログラムを走らせると、画面に4つの四角いスプライトが表示される。左側にある赤い四角をカーソルキーで操作して、右側に並んでいる3つの四角にぶつかってみよう。それぞれ、青の四角はふつうのスプライトで、緑はCCビットが1のスプライト、黒はICビットが1のスプライトになっている。また、スプライトが衝突したときに、衝突判定をしているかの確認用に割りこみを設定している。それぞれ試し終わったら、スペースキーで終了する。付録ディスクには、"C−SPRITE．SB2"というファイル名で収録してある。

ふつうのスプライトの場合

重なっているあいだはビープ音が鳴る

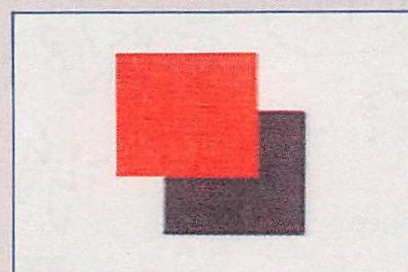
ふつうのスプライト同志が衝突したときは、行70で設定されたスプライト衝突割りこみがかかり、行100にあるサブルーチンが実行される。ここではたんにビープ音が鳴るだけだが、衝突割りこみが有効であることを確認しておいてほしい。

```
70 ONSPRITEGOSUB100:X=70:Y=90
100 SPRITEOFF:BEEP:RETURN
```

CCビットが1の場合

光と光を重ねたような感じになる

CCビットが1のスプライトは横にCCビットが0でスプライト面番号の小さいスプライトが表示されている部分だけ表示される。また、重なった部分の色が変化して、赤と緑のORの色、つまり黄色(色コード10)になるのがよくわかるだろう。

動かしているスプライトの色	1000	8
そこにあったスプライトの色 OR)	0010	2
重なった部分のスプライトの色	1010	10

ICビットが1の場合

写真ではわからないが、衝突判定しないのでビープ音が鳴らないのだ

右下の黒いスプライトにぶつかってみると、見た目では右上にある青いスプライトにぶつかったときとおなじだ。しかし、スプライト同志の衝突をしらせるはずのビープ音がまったく鳴らないだろう。このように、ICビットを1にすると衝突判定を無視させる働きがあるのだ。

COLORSPRITEの機能

COLORSPRITE命令で設定できる機能ビットには、全部で3種類があることはわかっただろうか。このなかで、もっとも派手な機能を持つ、CCビットについて紹介しておこう。

■CCビットの持つ機能

CCビットを1にすると、そのスプライト面は優先順位がなくなってしまい、そのスプライト面よりも優先順位の高いスプライトと、Y座標がおなじ部分だけが画面に表示されるのだ。

ところで、優先順位とはなんだろう？　案外知らない人もいるようだが、スプライトが横にいくつも並んで表示されると、スプライトモード1なら4つ、スプライトモード2では8つまでが表示される。このとき、スプライト面番号が小さい順に選ばれていくのだ。また、ふつうのスプライトが重なったとき、スプライト面番号の小さいほうはちゃんと表示されるだろう。これが優先順位で、つまり優先順位とはスプライト面番号の小ささのことをいうのだ。

CCビットが1のスプライトは、自分がもともと持っていた優先順位を放棄して、自分より高い優先順位のスプライトのところに居候(いそうろう)してしまうのだ。だから色が合わさってORの色になったり、居候されたスプライトを動かすと、一緒に動いてきたりするのだ。

■COLORSPRITE命令での多色刷り

ところで、COLORSPRITE命令には、機能ビットの設定のほかに、もともとスプライト面の色を定義するという機能を持っている。1枚のスプライトに何色も使えたりするのは、この命令で横1列ごとに色を定義しているおかげなのだ。

横1列ごとの色設定とは、そのスプライト面に表示されるスプライトの、上から横1列ごとということで、8×8ドットのスプライトなら8文字ぶん、16×16ドットなら16文字分のデータが必要になる。

今回掲載した2本のリストでは、どちらもスプライト面全体に対する設定だったので、横1列ごとの色設定の参考にはならないかもしれないが、機能ビットのCCビットとICビットの働きはよく見ておいてほしい。

次回の講座では、ちゃんと形のあるスプライトパターンを使って、COLORSPRITE命令とCCビットによる色付けなどを紹介したいと思う。

スーパー ビギナーズ

超初心者 Super Beginners' 講座

第15回 グラフィック入門その3

MSXFAN 付録ディスク
→P116の「スーパー付録ディスクの使い方」参照

今回はスプライトモード2の機能を使った、ちょっとしたシューティングをディスクに収録しておいた。ただし、あくまでサンプルなので、期待しないように。

多色スプライトの作成方法

スプライトモード2があまり使われていない理由のひとつに、そのスプライトパターンと色のデータの作りにくさがあげられるだろう。1枚のスプライトで、ドットの横1列ごとに色を付けて使うぶんには、そんなに大変な作業ではないのだが、複数のスプライトを重ね合わせ、何色もの色を使おうとすると、ツール、とくにエディタなどがなくては、とても作れたものではない。なぜなら、重ね合わせるスプライトパターンのそれぞれで、ドットがある、ないでたがいに干渉しあって色が決まるから、グラフィックでパターンを作るよりも、もっと大変な作業になるのだ。

■1枚のスプライトでの色付け

複数のスプライトを重ね合わせたパターンを作るのは大変な作業だが、1枚のスプライトに色付けする作業はかんたんだ。

なにしろパターンは1枚しか使わないから、スプライトモード1でスプライトパターンを作るのと、労力は変わらない。どこがちがうかといえば、表示する画面がSCREEN4以降の画面であることと、色のパターンデータが必要なことだけだ。

■リスト1「ヨット」

※ディスクに収録。LIST-1. SB3

```
10 DEFINTA-Z:SCREEN5,2:S=-1
20 READ A$,B$:IFA$="¥"THEN60ELSES=S+1
30 FORJ=0TO31:S$=S$+CHR$(VAL("&H"+MID$(A
$,J*2+1,2))):NEXT:SPRITE$(S)=S$:S$=""
40 IFLEN(B$)<4THENCOLORSPRITE(S)=VAL(B$)
:GOTO20
50 FORJ=0TO15:C$=C$+CHR$(VAL("&H"+MID$(B
$,J*2+1,2))):NEXT:COLORSPRITE$(S)=C$:C$=
"":GOTO20
60 FORI=0TOS:PUTSPRITE I,(100,50),,I:NEX
T:BEEP
70 IFSTRIG(0)=0THEN70ELSEEND
1000 ' ｻﾝﾌﾟﾙﾊﾟﾀｰﾝ 「ﾖｯﾄ」
1010 DATA 0000000202020606060606060201FF7F
1F8080C0C0E0E0F0F0F8F8FCF080FEFEFC
1020 DATA 0F0F0F0F0F0F0F0F0F0F0F0F0B0808
08
9999 DATA ¥,¥
```

下の図1を見てほしい。1枚のスプライトでの色付けサンプルとして、ヨットのパターンを掲載してみた。図のⒶのパターンは、ヨットのスプライトを単色で表示したもので、それにカラーコードの部分で示してある色を付けたものがⒷにあたる。

上のリスト1をRUNすると画面中央付近にヨットが表示されるだろう。リストの行1010がスプライトパターンデータ、行1020が色データになっている。

スプライトパターンデータは、

■図1 ラインごとの着色例

Ⓐ着色前のスプライトパターン

頭のなかでどんな色を付けるか想像しながら、スプライトモード1のパターンと同様にデータを作っていく。紙などに書くとよい。

Ⓑ着色後のスプライトパターン

1枚のスプライトでの色付けでは、横1列ごとに1色が限度。作ったパターンに色を付けていき、その色をカラーコードに変えていく。

カラーコード

……15(白)	:ØF
……15(白)	:ØF
……15(白)	:ØF
……15(白)	:ØF
……15(白)	:ØF
……15(白)	:ØF
……15(白)	:ØF
……15(白)	:ØF
……15(白)	:ØF
……15(白)	:ØF
……15(白)	:ØF
……15(白)	:ØF
……11(黄色)	:ØB
……8(赤)	:Ø8
……8(赤)	:Ø8
……8(赤)	:Ø8

自機の表示と操作

先月の講座では、カラーデータにある機能ビットのうち、CCビットとICビットについて紹介した。今回はそれらの機能ビットについて、付録ディスクに収録された、かんたんなシューティングゲーム(ファイル名：SHOOTING.SB3)をもとに紹介しよう。

■基本部分の作成と機能ビット

シューティングゲームなのだから、自機が必要だ。せっかくスプライトモード2を使うのだから、自機のパターンはスプライトを3枚重ねて立体感のあるものにした(P37参照)。

自機のパターンができたら、リスト4のように、試しに移動と表示の部分を作ってみよう。行70が入力、移動、表示の処理を行っていて、入力にはSTICK関数を使用している。

ここで注意してほしいのは、PUTSPRITEで動かしているのは1枚のスプライトなのに3枚とも動いていること。ここではCCビットの性質を利用して、表示命令を省略しているのだ。CCビットが1のスプライトは、CCビットが0のスプライトにくっついて動く性質があった。つまりここで動かしているスプライトは、CCビットが0のスプライトだということだ。

自機の移動と表示がうまくいったら、自機の表示をもう少し凝ってみよう。入力によりパターンが切り換わるようにするのだ。リスト4にリスト5を追加・修正すると、移動方向により自機のパターンが切り換わるようになる。ここでもPUTSPRITEに注意してみよう。自機を表示するスプライト面はおなじだが、表示するパターンの番号が入力により変化するのだ。

スプライトの色はスプライト面に定義される。だから自機パターンのように、おなじ色の組み合わせで表示されるパターンならば、スプライト面もおなじでかまわないのだ。

こんどはミサイルが発射できるようにしてみよう。リスト6を追加・修正すると、スペースキーでミサイルが発射できる。

このミサイルのパターンには赤い噴射の部分があり、この部分はぶつかっても熱いだけで爆発しないのがふつうだ。おなじスプライトで、一部だけ衝突判定しないようにするには、ICビットというのがあったのを覚えているかな？　このミサイルのパターンの赤い部分ではICビットを1にして衝突判定しないようにしているのだ。行1140のデータを見るとわかるだろう。

STICK関数とSTRIG関数

●書式：STICK(n)

STICK関数とは、カーソルキーやジョイスティックでの方向入力に使われるもので、上から時計回りに1～8、なにも入力がなければ0を値として持っている。

STICK関数のカッコの中に入れる数値(引数という)は0～2の範囲で、0ならカーソルキー、1ならポート1のジョイスティック、2ならポート2のジョイスティックの状態を調べてくれる。

ここでは、変数Sに値を保存(値がいつも変化しているので保存する必要がある)し、S＝3(入力が右)なら自機のX座標を加算、S＝7(入力が左)なら自機のX座標を減算しているのだ。

●書式：STRIG(n)

STRIG関数はスペースキーやジョイスティックのトリガーボタンの状態を調べる関数で、押されていれば－1、押されていなければ0を値として持っている。

引数の範囲は0～4で、0ならスペースキー、1と2はそれぞれポート1のジョイスティックのトリガーAとB、3と4はそれぞれポート2のジョイスティックのトリガーAとBの状態を調べる。

ここではミサイルの発射判定で入力があるかを調べている。

■リスト4 自機の表示と入力による移動

```
10 DEFINTA-Z:SCREEN5,2:S=-1
20 READ A$,B$:IFA$="¥"THEN60ELSES=S+1
30 FORJ=0TO31:S$=S$+CHR$(VAL("&H"+MID$(A
$,J*2+1,2))):NEXT:SPRITE$(S)=S$:S$=""
40 IFLEN(B$)<4THENCOLORSPRITE(S)=VAL(B$)
:GOTO20
50 FORJ=0TO15:C$=C$+CHR$(VAL("&H"+MID$(B
$,J*2+1,2))):NEXT:COLORSPRITE$(S)=C$:C$=
"":GOTO20
60 Z=120:P=0
70 S=STICK(0):Z=Z+(S=3)*(Z<240)*5-(S=7)*
(Z>0)*5:PUTSPRITE 0,(Z,190),,P:GOTO70
1000 ' ｼﾞｷ
1010 DATA 00000000010202020202020202020202
0A80404040C0C0C0D0D0D0D0D0D05050, 1
1020 DATA 0001010101040404041414347 4F436
0680C0C0C0C0E0E0F0F0F8F8FCFEFF3C30, 66
1030 DATA 01020202030101090909090909090908
0600000000800000000000000000000030, 76
9999 DATA ¥,¥
```

行10～50は前ページのリスト3とおなじものを使用している。この部分での役目は、スプライトパターンとスプライトの色を定義することだ。

行60では、自機のX座標用の変数Zを設定している。

行70は、このリストのメインルーチンの部分で、まず、STICK関数でカーソルキーの状態を調べ、それにより自機のX座標を計算し、自機のスプライトを表示している。表示しているスプライトは、CCビットが0のパターンだけだが、3枚全部が移動するのだ。

※付録ディスクに収録してあります。ファイル名：LIST－4. SB3

■リスト5 自機の傾きの表現

```
60 S(3)=3:S(7)=6:Z=120:P=0
70 S=STICK(0):Z=Z+(S=3)*(Z<240)*5-(S=7)*
(Z>0)*5:IFP<>S(S)THENP=S(S):PUTSPRITE1,(
Z,190),,P+1:PUTSPRITE2,(Z,190),,P+2
80 PUTSPRITE 0,(Z,190),,P:GOTO70
1040 ' ｼﾞｷ (ﾐｷﾞﾑｷ ﾉ ﾊﾟﾀｰﾝ)
1050 DATA 00000000010101010101010101010101
0580C04040C0000020202020303038B0A0, 1
1060 DATA 00000101020202020A0A0A1A3A7A1B
0380C0C0C0C040406060606070707 83060, 66
1070 DATA 01030202000404040404040404040404
030000000C080808080808080804060, 76
1080 ' ｼﾞｷ (ﾋﾀﾞﾘﾑｷ ﾉ ﾊﾟﾀｰﾝ)
1090 DATA 00000000000002020202020202020200
0580404000040404040505058585C5ED8A0, 1
1100 DATA 00000000030101010101010101010101
0380C0C0C080A0A0A0B0B0B8B8BCBE3860, 66
1110 DATA 0103030303000000404040C0C1C0E
030000000000000000000000000000060, 76
```

リスト4にこのリスト5を追加・修正すると、自機の移動方向により表示されるスプライトパターンが切り換わり、自機の傾きを表現できるようになる。自機の移動方向は、カーソルキーの入力が左か右かで決まるので、あらかじめそれぞれの方向に傾かせたパターンを作っておき、入力が変化したときに表示を切り換えるようにしている。配列変数S(n)と変数Pの使い方がポイントだ。ここでは3つの表示パターンとも、おなじ色データで表示できるので、表示するスプライト面番号を変えたり、色定義をやりなおさなくてもすむのだ。

※付録ディスクに収録されている、LIST－4. SB3をロードしてMERGE"LIST－5. SB3"を実行すると追加・修正できます。

■リスト6 ミサイルの発射と移動

```
60 S(3)=3:S(7)=6:X=0:Y=0:Z=120:P=0:Q=0
80 PUTSPRITE 0,(Z,190),,P
90 IFQTHENY=Y-8:PUTSPRITE9,(X,Y),,9:IFY<
-16THENQ=0
100 IFQ=0ANDSTRIG(0)THENQ=1:X=Z:Y=174:PU
TSPRITE9,(X,Y),,9
110 GOTO70
1120 ' ﾐｻｲﾙ -1
1130 DATA 0404040404040404040404040E150E15
04202020202020202020202070A870A820
1140 DATA 0F0F0F0F0F0F0F0F0B0E0F28282828
28
```

さらにこのリスト6を加えると、スペースキーでミサイルが発射できるようになる。行60にミサイルの座標と発射フラグの設定を追加し、行90でミサイルが発射されていれば移動・表示し、画面外に出たかの判定をしている。行100ではSTRIG関数でミサイル発射判定をし、発射するなら座標などを設定し表示する。

ミサイルの色データで、噴射部分はICビットを1にして衝突判定しないようにしてあるのがポイント。

※リスト5の部分に記された方法でリスト4と5を合わせたものに、MERGE"LIST－6. SB3とすれば追加・修正できます。

衝突判定などの工夫

スプライトを登場キャラクタをして扱う場合、とくにスプライトモード2ではいろいろな工夫をすることができる。

ミサイルの噴射部分は衝突判定をしないとか、もっと基本的な部分では、キャラクタとキャラクタがぶつかったときに割りこみを発生させてくれること。

これは、とくにシューティングゲームなどでは有効で、文字やグラフィックを使ったキャラクタでは得られない特典だ。

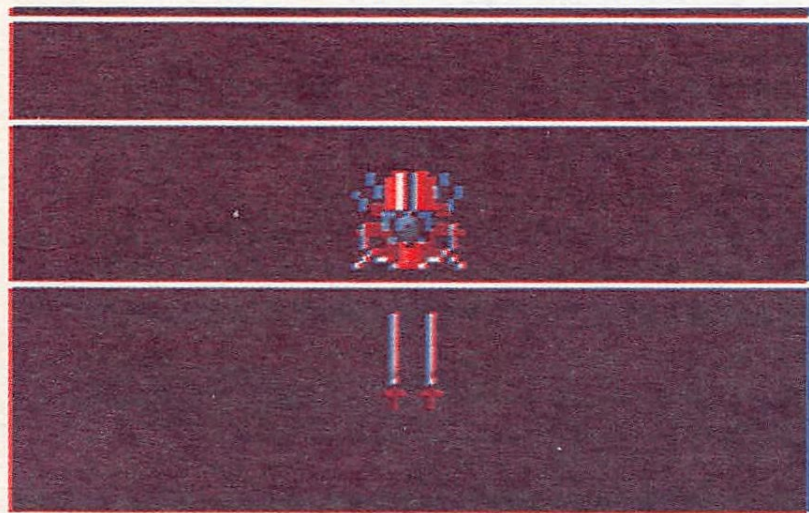
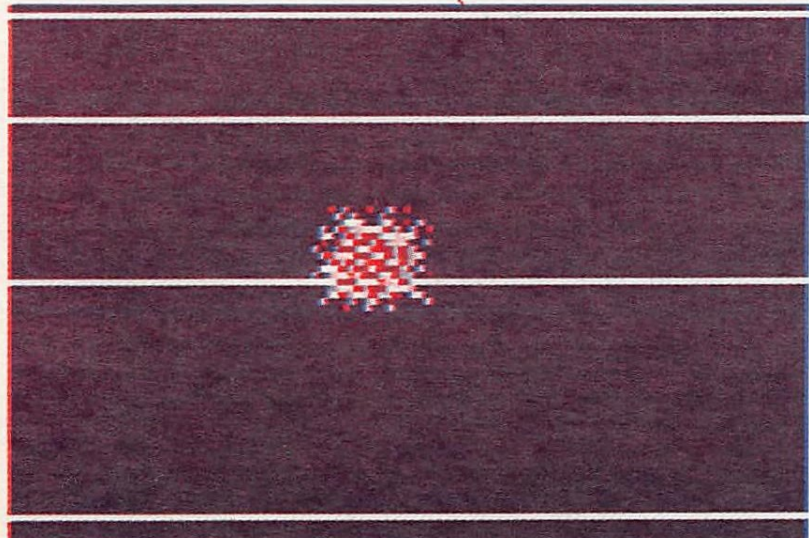
■衝突判定とは

スプライトの衝突判定による割りこみとは、リスト7のようにあらかじめおまじないをしておけば、スプライトとスプライトが重なったときに、特定の行を呼び出してくれるというものだ。だから、いちいち座標を調べなくてもいいし、ちゃんとドットとドットが重ならないと割りこみはかからない。

割りこみがかかったときに呼び出される行(割りこみサブという)では、リスト8のようにどのスプライトとどのスプライトが衝突したか判定したり、状況によっての処理を行う。今回のシューティングゲームでは衝突は2つのパターンしか起こらない。自機と敵の衝突か、もしくは敵とミサイルの衝突だ。

リスト9のように処理を設け、ゲーム中にICビットを1にしたり0にしたりすると、無敵などの効果が得られる。ICビットが1のときは、そのスプライトとほかのスプライトが重なっても、衝突判定を行わないので割りこみがかからないのだ。

敵が爆発すると、爆発のパターンになるが、このゲームではただの煙の固まりと考えているので、このスプライトには全体にICビットが1になっていて、自機やミサイルが重なってもなにも起こらない。

あまりおもしろいゲームにはならなかったが、敵や自機、ミサイルのパターンをオリジナルのものに変えたり、自分なりの改良を加えてみてほしい。

⬅敵とミサイルがぶつかる寸前の写真。ちょっと見た目ではなんかの甲虫に似ているが、いちおう敵の宇宙船という設定になっているのだ

⬅敵とミサイルがぶつかった直後の写真。敵が爆発のパターンに変わり、ミサイルは消えた。これに自機やミサイルが当たってもなにも起こらない

■リスト7 スプライト衝突割りこみの定義

```
50 S(3)=3:S(7)=6:X=0:Y=0:Z=120:P=0:Q=0:X
X=0:YY=-20:ONSPRITEGOSUB160:GOTO150
```

これはSHOOTING. SB3のリストの一部。この行の終わりにあるON SPRITE GOSUB～というのが衝突割りこみを定義している部分。SPRITE ONという命令を実行すると、衝突割りこみがかかるようになり、～の部分に書かれた行を呼び出すようになる。

■リスト8 スプライト衝突割りこみ処理

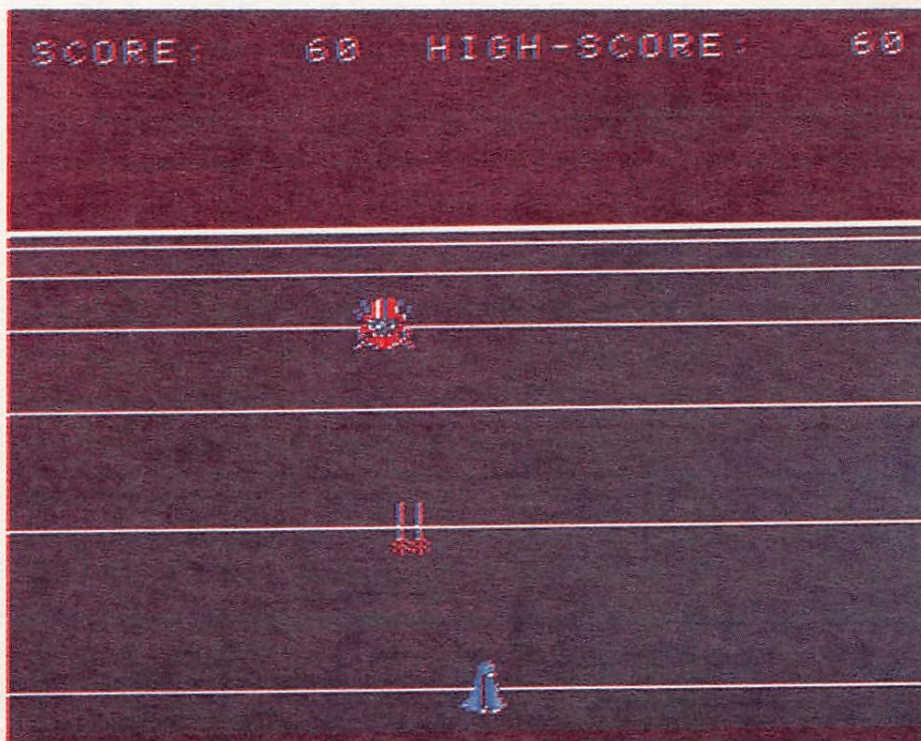
```
160 SPRITEOFF:COLORSPRITE(3)=33:IFY<175T
HENM=X:N=Y:PUTSPRITE7,(0,217):Q=0ELSEM=Z
:N=190
170 FORJ=3TO6:PUTSPRITEJ,(M,N),,J+12:NEX
T:IFN=190THEN180ELSEV=0:N=8:T=15:U=2:SC=
SC+10:IFHS<SCTHENHS=SC:GOTO200ELSE200
180 FORI=0TO2:PUTSPRITEI,(0,217):NEXT:LI
NE(90,98)-(167,109),8,BF:LINE(89,97)-(16
8,110),15,B:COLOR10,0:FORI=93TO94:PRESET
(I,100),,TPSET:PRINT#1,"GAME OVER":NEXT
190 IFSTICK(0)=1THEN40ELSE190
200 PRESET(10,2):PRINT#1,USING"SCORE:###
##  HIGH-SCORE:#####";SC;HS:RETURN
```

このリストの最初にあるSPRITE OFFという命令で割りこみが解除される。つぎにあるCOLORSPRITE命令で、爆発パターンのICビットを1にしているのだ。このゲームでは、自機と敵かミサイルと敵しか衝突しない。敵が自機とおなじY座標なら(変数Yは敵のY座標)自機と敵、そうでなければミサイルと敵が衝突したとわかるのだ。衝突があった位置に爆発パターンを表示し、敵と自機が衝突していたなら行180以降のゲームオーバー処理を実行する。敵とミサイルならスコアを加算し、もとの処理にもどる。衝突割りこみ処理はサブルーチンとして呼び出されるので、RETURNでもとの処理にもどる。

■リスト9 無敵などの工夫

```
100 C=CMOD7+1:COLOR=(8,C,0,0):IFGTHENG=G
-1:IFGTHEN120ELSECOLORSPRITE(0)=1:COLOR,
,1
110 IFPEEK(-1045)=251ANDSC>49THENCOLORSP
RITE(0)=33:COLOR,,2:SC=SC-50:G=9:GOSUB20
0
```

行100では敵やミサイルに使われているカラーコード8の色をパレット切り換えして点滅させている。

このゲームではスコアが50以上あれば、GRAPHキーを押したあと少しのあいだ無敵になる。ただし、スコアは−50される。パレット切り換えの次からがその無敵の処理が行われている部分で、変数Gは無敵になっている時間。0になると、自機の1枚目のスプライト面の色をICビットが0の状態に設定しなおして、周辺色を黒にもどしている。行110ではGRAPHキーの入力判定を行い、判定があれば自機の1枚目のスプライト面の色をICビットを1にして設定し、無敵になったことを示す周辺色を緑色に変えている。

【SHOOTING. SB3の遊び方】カーソルキーの左右で自機を移動し、画面上から迫ってくる敵をよけ、スペースキーでミサイルを発射し、敵を撃破するゲーム。スコア50と引換えに、GRAPHキーで一瞬無敵になれる。リプレイは上キー。

スーパー

超初心者 Super Beginners' 講座

ビギナーズ

第16回

グラフィック入門その4

スプライトモード2にはさまざまな魅力がある。しかしそのぶん、データの作成や設定が複雑になっているのできちんと把握していないと思うようにはいかないのだ。

スプライトモード2のまとめ

今月はスプライトモード2についての解説の締めくくりの意味で、紹介していなかったECビットに関する内容も含めた、スプライトモード2のまとめをしようと思う。

■スプライトモード2とは

MSX2以降の機種で、画面モードをSCREEN4以降に指定すると、自動的にスプライトモード2の状態になる。

スプライトモード2の特徴をかんたんにまとめてみると、

①横に4枚までしか並べられなかったスプライトが、スプライトモード2では8枚まで可能

②COLORSPRITE命令を使用することで、パターンのドット横1列ごとに色を設定できる

③COLORSPRITE命令で設定するデータは、パレットコード(カラーコード)のほかに機能ビットの設定もでき、スプライトの重ね合わせや、衝突判定をするかの設定もできる

以上の3つが大きな特徴で、スプライトモード1とくらべて1枚のスプライトでできる内容が多くなり、スプライトの組み合わせに対する処理も種類があっておもしろい。

■COLORSPRITEの書式と設定するデータ

これらの特徴は、あらかじめCOLORSPRITE命令を使い、各スプライト面に対してデータの設定をしていなければ利用できない。

スプライト面に対するデータの設定方法には2通りの方法があり、1つはスプライト面全体に対して数値データで設定する方法。そしてもう1つはスプライト面のドット横1列ごとに文字列データで設定する方法だ。

それぞれの方法で設定されるデータは、パレットコードを基本とし、各機能ビットを1にするときに対応する数値を加えたものがデータとなり、スプライト面全体に対する設定ならば、このまま数値で設定できる。

文字列データで、スプライト面の横1列ごとに設定するときは、各列ごとに数値を計算し、それを文字コードとする文字に変換する。たんにCHR$関数の引数にして、そのまま使用してもいい。そして、それらの文字を上から順に並べたものを、COLORSPRITE$に設定すればいいのだ。ただし注意してほしいのは、数値データを文字列データにするやり方は、スプライトパターンデータとおなじだが、指定する文字数は縦のドット数までということと、データは上書きされるので、少ない文字数で設定したときは、残りの部分に今までのデータが残ってしまうという点に注意。

●スプライトモード1

- ・SCREEN1～3ではスプライトモード1が選択される
- ・横に4つまで同時表示できる

●スプライトモード2

- ・SCREEN4以降ではスプライトモード2が選択される
- ・横に8つまで同時表示できる
- ・COLORSPRITE命令により各スプライト面ごとに色パターンを定義することができ、同時に各機能ビットを設定することで、そのスプライト面に表示されるスプライトの衝突判定の有無や重ね合わせ処理などが設定できる

●COLORSPRITEの書式

・COLORSPRITE(n)=数値データ

スプライト面nに表示されるスプライトの色を、指定した数値の色に定義する。同時に各機能ビットの設定も行う

・COLORSPRITE$(n)=文字列データ

スプライト面nに表示されるスプライトの色を、横1列ごとに定義する。同時に各列ごとに機能ビットの設定も行う

●COLORSPRITEに設定するデータ

0～15……パレットコード
+32………ICビットをon(衝突判定なし)
+64………CCビットをon(優先順位なし)

※パレットコードの数値にそれぞれの機能ビットを設定値をたしたものをCOLORSPRITEの数値データとして設定する。文字列データの場合は、CHR$関数内のキャラクタコードとして使用する。なお、ECビットは+128だが、文字列データでのみ使用可能。

SB進路相談

今月から、この欄外のスペースを使って「スーパービギナーズ講座・進路相談」と題して、SBの読者からよせられた質問に答えていこうと思う。とはいうものの、今月は、はじめましてのあいさつと、進路相談であつかう質問の種類を紹介することにしよう。【進路相談で受け付ける質問】この進路相談で受け付ける質問は、①いままでのスーパービギナーズ講座の記事で、わからなかったところや、こんなところも知りたいという質問や要望　②この命令の使い方

各機能ビットの働き

スプライトモード1では、1つのスプライトにつき1色しか使えず、おまけに横に4枚までのスプライトしか表示できなかったので、どんなにがんばっても4枚重ねて4色しか表現できなかった。しかし、スプライトモード2では、1枚のスプライトでも、ドットの横1列ごとに色を指定できるので、8×8ドットの1枚のスプライトでも、最大で8色使えるようになった。

さらに機能ビットを操作することで、ドットの横1列で2枚重ねなら3色、3枚なら7色、4枚重ねれば15色すべてを同時に使うこともできるのだ。

機能ビットには、このようにそのスプライト面に表示されるスプライトの性質に関する設定を行う働きがあり、ECビット、CCビット、ICビットの3種類が用意されている。

これら機能ビットの働きは、あとで詳しく説明するが、ECビットは表示されるスプライトの位置を左に32ドットずらす働きがあり、CCビットはそのスプライト面の優先順位を放棄させる働きがある。そしてICビットには、そのスプライトがほかのスプライトと衝突したとき、衝突判定をするかどうかを設定する働きがあるのだ。

ECビット　32ドット表示位置をずらす

このECビットが1のとき、スプライトの表示位置を左に32ドットずらす働きがある。これはスプライトモード1と共通の機能ビットで、スプライトを画面の左端から出現させるときなどに使用する。

PUTSPRITE命令でスプライトを表示させるとき、X座標をマイナスの値にすると、自動的にECビットが1になり、VRAMにあるX座標の値が調整(+32)される仕組みになっているので、画面の左端からスプライトを表示させるのにECビットが関係していると気付かない人も多いだろう。

しかし、このECビットは、COLORSPRITE命令に数値で設定するとエラーになるので文字列でしか設定できず、PUTSPRITEで座標を指定すると、それにともない変化してしまうので使いにくい。

だが、スプライトの一部だけ32ドット左にずらすというおもしろいことができるので、使いようによっては貴重な機能だ。

ICビット　スプライト衝突判定の設定

ICビットの働きは、ほかのスプライトと重なったときに、衝突衝突を発生するかどうかを設定するものだ。

つまり、ICビットを1に設定したスプライトは、衝突判定を行わなくなるので、スプライト衝突割りこみを使用したプログラムでも、そのスプライトとほかのスプライトが衝突しても割りこみ処理を実行しない。

例えば、上から石と雪のかたまりが降ってくるのをよけるゲームを作るとき、雪のかたまりは痛いけどゲームが続行できるなんていうのにちょうどよい。

また、文字列データで設定するときは、部分的に設定することができるので、例えば風船のヒモの部分は衝突判定しないがゴムの部分は衝突判定する、というような工夫もできる。

ICビットは細部の工夫に利用すると効果的だろう。

CCビット　スプライト面の優先順位をなくす

●基本的な性質

CCビットには基本的な性質として、以下の3つがある。

①CCビットが1のスプライトは優先順位を放棄し、基本的に優先順位は持たない

優先順位とは、スプライトが2枚以上重なったとき、どのスプライトから優先的に表示するかの順位で、スプライト面番号が小さいほど高くなる。

②ICビットとおなじく、CCビットが1のスプライトも衝突判定を行わない

③CCビットが1のスプライトが持っていた、もとの優先順位より高い優先順位を持つスプライトが存在するとき、そのスプライトと等しい優先順位を得ようとする。もし、条件にあうスプライトが複数存在するときは、もとの優先順位にいちばん近いスプライトと等しい優先順位を得ようとする

ここで注意してほしいのは、もとの優先順位より高いスプライトでも、おなじようにCCビットが1のスプライトは優先順位を放棄しているので、この条件の対象にならないということ。

●表示されるための条件と表示されたときの性質

CCビットが1のスプライトは単独では画面に表示されず、表示するためには条件を満たさなくてはいけない。また、その性質も条件を満たすまでは基本的な性質のままなのだが、条件が満たされたあとでは性質が変化してしまうのだ。

④CCビットが1のスプライトは、③の条件にあうスプライトとY座標が等しい部分のみ画面に表示される。このとき、③の条件にあうスプライトとおなじ優先順位を得ることができる

CCビットが1のスプライトが表示されるための条件がこれで、横1列ごとに③の条件にあうスプライトがあるかの判別をする。条件にあうスプライトがあれば、その列は画面に表示されて優先順位を得る。つまり、CCビットが1のスプライトが表示されたときは、必ず優先順位を得た状態になっている。

またこのことからスプライト面0のCCビットを1にしても、③の条件にあうスプライトは存在しないので、絶対に画面に表示されないことがわかる。

⑤CCビットが1のスプライトが優先順位を得た場合、そのスプライトの性質も受け継ぐ

CCビットが1のスプライトが表示されたときに得るのは優先順位だけでなく、ICビットの状態も得ることができるのだ。

⑥CCビットが1のスプライトと、③の条件にあうスプライトとが重なると、両方のスプライトの色のORの色になる

④の性質から、CCビットが1のスプライトは、③の条件にあうスプライトと重なったとき、等しい優先順位を持っている。

優先順位が等しいスプライト同志が重なったとき、その部分の色は、たがいに隠しも隠されもせずに、両方の色のORの色となって表示されるのだ。

これがCCビットの最大の魅力となる部分で、この性質からスプライトを重ね合わせての色の変化が楽しめるのだ。

⑦CCビットが1のスプライトは、③の条件にあうスプライトが表示されると、おなじ位置に移動して優先順位を得る

これは③の性質の強さの現れのようなもので、これを利用すれば重なったスプライトを1つのパターンとして操作することが可能になるのだ。

ほかの機能ビットに比べて、このCCビットはかなり複雑だ。しかし、CCビットはスプライトモード2の目玉的な存在なので、理解できなくても、どんどん使ってみてほしい。

SB進路相談

や意味が知りたい　③ファンダムのプログラムでこんな計算式(または命令、データなど)があったけど、どんな意味があるのか知りたい　④スーパービギナーズ講座で紹介してほしいテーマ　以上の4つに限ることにする。でも、その他の内容でも、ハガキ1枚むだにしてもいいという覚悟があるなら、どんどん送ってきてもかまわない。もしかしたら、答えることができるかもしれない。

超初心者 Super Beginners' 講座（スーパービギナーズ講座）

第17回 グラフィック入門その5

付録ディスク
→P116の「スーパー付録ディスクの使い方」参照

MSX1はVRAMが16Kしかなかったのに、なぜ、MSX2以降の機種ではVRAMがどーんと増えたか知っているかな？　もっと自由に色が使えるように、SCREEN5以降の画面が用意されたからなんだ。

グラフィックとの接し方

今回は、「それほどCGなどに興味はないが、MSXで絵を描くとは、いったいどんなものなのだろう」という、ほとんどグラフィック未体験者のために、グラフィック画面との接し方について、紹介しよう。

ただし、SB講座はCG講座ではないので、うまいグラフィックの描き方などは教えられない。あくまで、自分の手でプログラムを組んで、グラフィック画面を使ってみるための手ほどきにすぎない。そもそもわたし自身、CGは苦手なほうなのだ。

■グラフィック画面のいろいろ

グラフィック画面とひとくちにいっても、それぞれ特徴を持った画面がいくつもある。

そのなかでも、SCREEN2～4までの画面と、SCREEN5以降の画面では、大きな差があるのだ。

とはいうものの、いったいどれくらいの差があるのだろうか。

まず、SCREEN2～4では使えないが、SCREEN5以降なら使える命令がある。

グラフィックの複写や重ね合わせなどで威力を発揮するCOPY命令や、複数の画面を扱うことができるSETPAGE命令などが、まずあげられる。

また、命令自体はあるが、その機能に差があるものがある。

PAINT命令で塗りつぶすとき、SCREEN2～4では枠と同じ色でしか塗ることができなかったが、SCREEN5以降では違う色で塗ることができる。

さらに、LINE命令などで、SCREEN5以降ではロジカルオペレーション（論理演算）を使うことができるので、グラフィックの反転や特定の色以外の塗りつぶしなどができておもしろい。

これだけでもSCREEN5以降のほうがグラフィックを扱うのに、いろいろできて便利なことがわかる。

■おすすめはSCREEN5

ところで、SCREEN5以降なら、どの画面でもグラフィックが扱いやすいかというと、そうでもないようだ。

MSX2では、SCREEN5～8の4つの画面が使えるが、それぞれの画面モードの特徴と使いやすさについて紹介しよう。

【SCREEN5】

特徴がないように思われるくらい標準的なグラフィック画面。

横256ドット×縦212ドットの画面構成で、色はパレット16色を自由に使うことができる。

ページは4（VRAM64Kでは2）ページ使用可能なので、アニメーションやCOPY命令を駆使したゲームなどでよく使われている。

SCREEN5以外の画面は扱いやすく、グラフィックエディタなどのツールも多い。

【SCREEN6】

特徴は、パレット16色のうち、たったの4色しか使えない反面、横のドット数が512ドットとふつうより細かくなっていること。グラフや文字を表示するのにはいいかもしれない。ページが4（VRAM64Kでは2）ページあるので、もしかしたら使い道があるかもしれない。

【SCREEN7】

この画面もSCREEN6とおなじく、横のドット数が512ドットと細かい。ただし、パレットは16色を自由に使える。

CGを描くのには向いている画面のようで、投稿の大半はこの画面のものだが、意外と対応しているグラフィックエディタは少ないようだ。

SCREEN6もそうだが、横のドット数が細かいので、RGB入力のモニターでも縦の線が細くて見づらいのが難点。

【SCREEN8】

特徴は256色が自由に使えること。色数が多いのはいいが、0～255の色コードと色との対応を覚えるのがめんどうだし、パレットもなく、スプライトの色が固定16色しか使えないなどちょっととっつきにくい。

SCREEN7と8は、VRAM64Kの機種ではふつうは使えない。また、ページはそれぞれ2ページずつ使用できる。

SCREEN5以外の画面は状況に合わせて使用したほうがいい。また、SCREEN6と7では横のドット数が倍になっているが、スプライトの座標は変わらないので注意が必要だ。

■グラフィックとの接し方

プログラムでグラフィックを扱うには、2つの方法がある。

1つは決められた絵を作成するプログラムで、ゲームの背景やAVフォーラムのビジュアル作品がそのたぐいだ。

もう1つはエディタなどで、グラフィックを作る手助けをするプログラムだ。

グラフィックに慣れるいちばんいい方法は、エディタを自作することだと思う。グラフィック命令の機能も覚えられるし、処理の組み方もいろいろあって力が付くので、ぜひ挑戦しよう。

■ちょっとした注意事項

グラフィック画面を扱うとき、注意してほしいことがある。

SCREEN命令で画面モードを設定すると、自動的にCLSが実行されること。CLSが実行されると、画面が周辺色で塗りつぶされるので、画面モードを変えてから周辺色を設定するときはCLSを実行しよう。

ページ1以降の画面を使うときもプログラムの始めでCLSをして初期化しよう。画面モードを変えただけではページ1以降はCLSされないからだ。

パレットを切り換える場合、画面モードが変わるとパレットが初期化されるので注意しよう。

SB進路相談

【3月号のフォロー】3月号で掲載した「MAKEDATA．BAS」の使用方法について、多くの質問電話や質問ハガキがあった。そのつど詳しく説明していたのでは大変なので、分かる人には分かる、という程度で答えていたが、やはりものがものなだけに、きちんと誌面で答えなければいけない。そこで今月のSB講座では、MAKEDATA．BASで利用できるグラフィックが作れる、サンプルリスト1を掲載した。やり方は、まず行100からのDATA

プログラムで絵を描く

プログラムで絵を描くというのは、文字通りプログラムリストにあるグラフィック命令によって、画面に絵を表示するということだ。だから、あらかじめ、どのような絵にするか決めておかなければうまくいかないし、どんな絵にするかによって、プログラム中で使用される命令も変わってくる。単純なパターンや背景を使うときは、プログラムで描いたほうがいい場合が多いのだ。

■リスト1　パターン作成プログラム

```
10 SCREEN5:COLOR15,0,0:CLS
20 FORY=0TO15:READ A$
30 FORX=0TO15
40 C$=MID$(A$,X+1,1)
50 C=VAL("&H"+C$)
60 PSET(X,Y),C
70 NEXT:NEXT
80 IFSTRIG(0)=0THEN80 ELSEEND
90 COPY(0,0)-(15,15)TO"SAVE     .GRP":END
100 DATA 0000088880000000
110 DATA 0000888888000000
120 DATA 0000888888800000
130 DATA 00008B8888808000
140 DATA 00008B4B88800800
150 DATA 00000B4BB8800800
160 DATA 00000BBB88088000
170 DATA 0000007770000000
180 DATA 0007447744000000
190 DATA 0007444444400000
200 DATA 0007444444400000
210 DATA 0007444444F00000
220 DATA 00007447FF400000
230 DATA 0000077444440000
240 DATA 0000CC44444C0000
250 DATA 0000CCC444C00000
```

■プログラム解説

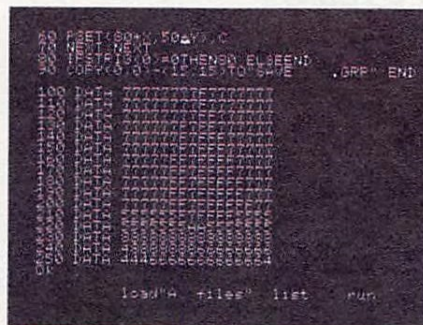
10　画面モード設定／画面の色設定／画面消去⇨画面が背景色(黒)になる
20　Y座標用ループ開始／パターンデータ読みこみ
30　X座標用ループ開始
40　読みこんだパターンデータを、X座標の増加とともに、左側から1文字ずつ取り出す
50　取り出した文字を数値に変換
60　指定座標に点を表示
70　X座標、Y座標のループ閉じ
80　スペースキーが押されるまでこの行を繰り返し、押されたら終了する
90　画面に表示されたグラフィックをディスクに保存して終了する
100〜250　パターンデータ

■使用方法

このリスト1は、あらかじめデータのところに示されたパターンを、画面に表示するプログラムだ。データは、16進形式の文字列データで、それぞれ1文字で1つのドットの色を表す。

プログラムで絵を描くには、この他にも、DRAW命令やLINE、CIRCLE、PAINTなどの命令もあり、これらはいずれ紹介しよう。

行80を削除すれば、作成したパターンをディスクに保存してくれる。ちなみに、保存したパターンは、3月号で付録ディスクに収録した「MAKEDATA. BAS」で使用できる。

※付録ディスクに収録されています
ファイル名SAMPLE−1. SB5

⬆行100以降のデータをいろいろ変えてみよう。たとえばこのようにすると……

⬆どこかでみたような絵だが、こんなパターンのグラフィックが表示されるのだ

ツールなどで絵を描く

プログラムではどうやっても描けそうもない絵、たとえばアドベンチャーゲームに出てくるCGなどは、とてもプログラムでは描けない。こういった絵は、グラフィックエディタなどのツールを使って描くのがふつうだろう。使う人から見れば、直接画面に絵を描くようなものだけど、プログラムから見れば間接的に絵を描いているわけだ。

■リスト2　簡易線画ツール

```
10 SCREEN5,0:COLOR15,0,0:CLS
20 SPRITE$(0)="みたタ♠":C=15
30 'on key gosub,,,,,,,,200,300:GOSUB120
40 A=PAD(12):XX=X:YY=Y
50 X=X+PAD(13):IFX<0THENX=0 ELSEIFX>255T
HENX=255
60 Y=Y+PAD(14):IFY<0THENY=0 ELSEIFY>211T
HENY=211
70 A=VAL("&H"+INKEY$):IFA>1THENC=A
80 PUTSPRITE 0,(X,Y),C,0
90 IFSTRIG(1)THENLINE(XX,YY)-(X,Y),C
100 IFSTRIG(3)THENCLS
110 PUTSPRITE 0,(X,Y),C,0:GOTO40
120 FORI=9TO10:KEY(I)ON:NEXT:RETURN
130 FORI=9TO10:KEY(I)OFF:NEXT:RETURN
200 ' F9 ｷｰ ｻﾝﾌﾟﾙ:ｶﾞﾒﾝ save ――――――――
210 GOSUB130:BEEP
220 BSAVE"SCREEN5 .GRP",0,&H69FF,S
230 GOSUB120:RETURN
300 'F10 ｷｰ ｻﾝﾌﾟﾙ:ｶﾞﾒﾝ load ――――――――
310 GOSUB130:BEEP
320 BLOAD"SCREEN5 .GRP",S
330 GOSUB120:RETURN
```

■プログラム解説

10　画面初期化
20　カーソルのスプライトパターン定義／表示色設定
40　マウスから座標データ読みこみ／カーソルの座標保存
50〜60　カーソルの座標計算・調整
70　キー入力／入力により表示色変更
80　カーソル表示
90　左ボタンが押されていれば、表示色で線を描く
100　右ボタンが押されていれば画面を消す
110　行40へ飛ぶ
30、100〜　下を参照

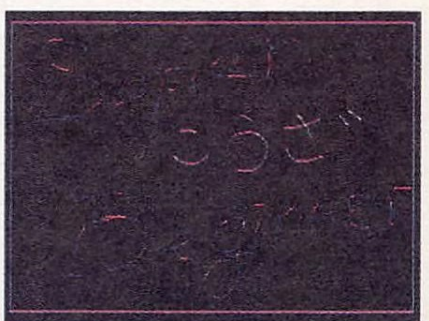
■操作方法

これはかんたんな線画ツールで、マウスで使用する。左ボタンで線を引き、右ボタンで画面を消す。2〜9とA〜Fを入力すると色を変更できる。

また、行30の「'」を消すと、F9キーでグラフィックをディスクに保存し、F10キーでディスクからグラフィックを読みこめるようになる。

F1キーなどに色コードを、KEY1,"C232C68986"などとして登録しておき、実行中にそのファンクションキーを押したまま、線を引くとおもしろいぞ。

こういったエディタなどのプログラムは、使う人が絵を描くための手助けをするものなので、間接的に絵を描いているといえるだろう。

※付録ディスクに収録されています
ファイル名SAMPLE−2. SB5

⬆ターボRの高速モードで描いたほうが、きれいな線が描ける

SB進路相談

文を、作成したいスプライトの色付きパターンデータに変える。データは16進数形式の文字列になっていて、それぞれ1文字が1つのドットの色に対応している。つぎに行80にある「END」を消して実行する。行80そのものを消してでもいい。すると行90にあるCOPY命令によって、作成したグラフィックが「SAVE. GRP」というファイル名でディスクに書きこまれるので、あとはMAKEDATA. BASのプログラムを実行すればOKだ。

SCREEN5のVRAM

VRAMなんて聞くと、うえ〜っと思う人も多いだろう。

わたしも過去にそういう経験がある。多色刷りはおろか、グラフィック画面に文字を表示する方法も知らなかったころ、VRAMという文字を見ると、読み飛ばしてしまっていた。

しかし、そのために、スプライトのゴーストや、パレットやグラフィックがうまくセーブされずに泣いた経験がある。

あーだこーだやって、結局わからなくて投げ出したプログラムがいくつあったことだろう。

VRAMをよく理解していれば、そんなくやしい思いをしなくて済んでいたのは事実なのだ。

■グラフィックの保存

画面に描いたグラフィックをディスクに保存するとき、方法は2つある。

1つはCOPY命令を使って行う方法で、前ページのリスト1の行90のような使い方をする。

始めに指定している2つの座標は、LINE命令とおなじように、その座標を結ぶ線を対角線とする長方形の部分を、ファイルとして保存するのだ。

もう1つの方法はBSAVEを使ったやり方で、リスト2の行220のように、VRAMのアドレスで指定する。

BSAVEで保存する場合、パレットやスプライトパターンなども一緒に保存することができるので、パレットを切り換えていたり、スプライトパターンも保存したいときには便利だ。

このアドレス指定を面倒臭がってはいけない。なんでも全部保存すればいいと思っていると、わたしとおなじ目に会う可能性がある。

SCREEN5のVRAMについては下で詳しく説明しているので、必要な部分がきちんと保存されるように、アドレスをよく調べて指定しよう。

■SCREEN5のVRAM構成

SCREEN5のVRAMは、各ページごとに下で紹介している5つの領域で構成されている。また、初期状態ではページ0の先頭アドレスが0になっているが、SETPAGE命令でアクティブページを切り換えたあとでは、アドレス0はそのページの先頭アドレスに変化しているので注意しよう。

各テーブルの説明

●パターン名称テーブル

●アドレス：&H0000〜&H69FF
●内容　：画面の各ドットのカラーコードを保存している領域
●説明　：この領域が、画面に表示されるグラフィックそのものを保存しているところだと思っていい。だから、BSAVE命令でグラフィックをディスクに保存するときは、このアドレスの部分を指定すればOKだ。データの内容は右図のように、1バイトで横に並んだ2ドットを表し、画面左上の位置から右へ、右端までいくと1ドット下へ、というように並んでいる。

●スプライトカラーテーブル

●アドレス：&H7400〜&H75FF
●内容　：各スプライト面の1ラインごとの色データを保存している領域
●説明　：前回まで紹介していたスプライトモード2で使用される領域で、COLORSPRITE$で設定したデータの内容が保存される。スプライト面0〜31のそれぞれに対し、16バイトずつ割り当てられていて、スプライト面0のものから順に並んでいる。8×8ドットのスプライトを使用していたり、スプライト面全体に設定しても、16バイトの割り当ては変わらない。

●スプライト属性テーブル

●アドレス：&H7600〜&H767F
●内容　：各スプライト面に表示されるスプライトの情報を保存する領域
●説明　：各スプライト面ごとに、4バイトずつ割り当てられており、表示されるスプライトのパターンの番号や、その座標が保存される。4バイト目の未使用部分は、スプライトモード1のときはカラーコードとなっているが、スプライトモード2では、前述のスプライトカラーテーブルで色の情報を管理しているので使用されない。その他の部分はスプライトモード1と同様の働きをしている。

●パレットテーブル

●アドレス：&H7680〜&H769F
●内容　：各パレットの成分の情報が保存されている領域
●説明　：パレットコード0〜15のそれぞれにつき、2バイトずつ割り当てられている。データの内容は右図のように、1バイト目は赤と青の輝度、2バイト目が緑の輝度となる。パレットを切り換えて作成したグラフィックを保存する場合、このアドレスの内容も含めて保存しておけば楽だろう。ただしこの領域はあくまで内容が保存されているだけなので、パレットをこの領域の内容で切り換えたいときは、COLOR=RESTOREを実行する必要がある。

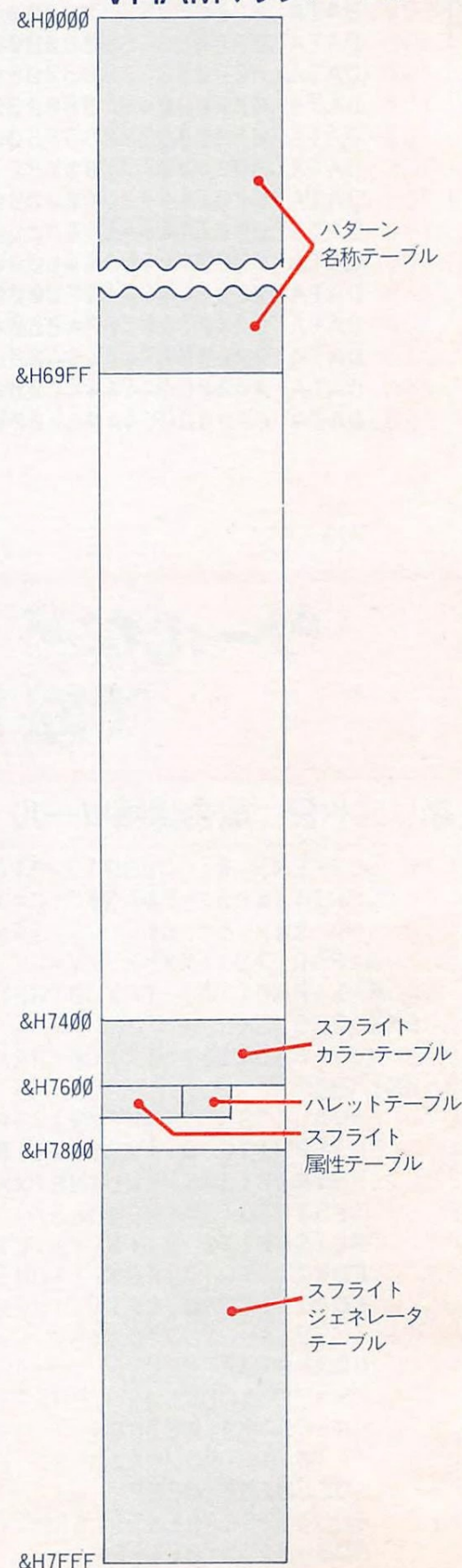
●スプライトジェネレータテーブル

●アドレス：&H7800〜&H7FFF
●内容　：各スプライトのパターンデータが保存されている領域
●説明　：この内容はほかの画面モードのときと同じで、各スプライトのパターンデータが保存されている。データは8×8ドットのスプライトなら、8バイトずつ256パターンぶん、16×16ドットのスプライトなら、32バイトずつ64パターンぶんが保存される。この領域に限らないが、これらの領域は各ページごとにあるので、おなじスプライトを別のページで使用するときなどは、そのときどきでスプライトパターン定義をする必要があるので注意しよう。

SB進路相談 【SB進路相談への質問、受け付け中】このコーナーではSBの諸君が抱いている素朴な質問に答えていこうと思っている。「今月の記事の××がわからない」とか、「この命令はどういう意味か？」など、頭を抱えるくらいなら、SB進路相談まで送ってみよう。ただしハガキが届いてから誌面で答えられるまで少し時間があくので、出しっぱなしにしないで、手当たり次第に調べてみよう。それがSB脱出への第一歩になるからだ。

超初心者 スーパー Super ビギナーズ Beginners' 講座

第18回
グラフィック入門その6

付録ディスク
→P116の「スーパー付録ディスクの使い方」参照

今月から、いよいよグラフィック命令を紹介していこうと思う。ところで、先月の簡易線画ツールをやってみたかな？　SB講座では、グラフィック命令の紹介とともに、拡張の方法も紹介していこうと思う。

ワクの中の色塗り命令

先月の簡易線画ツールを実行してみてくれたかな？　あれはマウスで線を描くだけのかんたんな内容のものだった。だから、ちょっと絵を描いてみようと思っても、なかなかうまくは描けなかっただろう。なにしろ色を塗ることさえ容易ではなかったのだから。

そこで、今回はワクの中を塗る命令「PAINT命令」を紹介しよう。さらに、紹介したグラフィック命令を、簡易線画ツールに組みこむ方法も紹介するので、そのうち、ある程度使えるツールになればと思っている。

■ワクの中を塗る命令

PAINT命令は、ワクの中を色で塗る命令だ。書式は右上にあるように、塗りはじめる座標と塗る色、塗るときに何色のワクの中を塗るか（境界色）の4つの情報を指定する。

すぐ下のリストはワクのサンプルで、CIRCLE命令を使っている。CIRCLE命令については今回は紹介しないが、円を描くためのグラフィック命令とだけいっておこう。

このサンプルをRUNすると、その下にある写真のように、色の違う3つの円が表示されるので、この円の中をPAINT命令を使って塗ってみよう。

下の3つの写真はその例だ。塗りはじめる座標は、3つとも円の中心となっているが、塗る色と境界色が少しずつ違う。そのために、内側の円が塗りつぶされて消えてしまったりしているのがわかるかな？

PAINT命令で色を塗るときに境界色の指定を間違うと、塗ってほしくない部分まで塗られてしまうことがある。

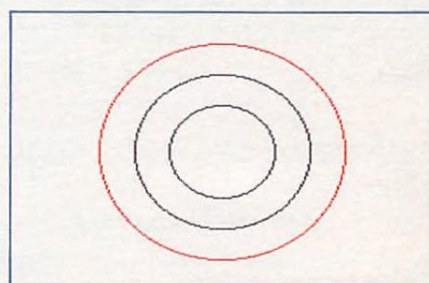
プログラムで絵を描くときはやり直しができるけど、ツールなどで絵を描くときは、いちどセーブなどをしておいたほうがミスしたときに助かるだろう。

■PAINT命令の書式

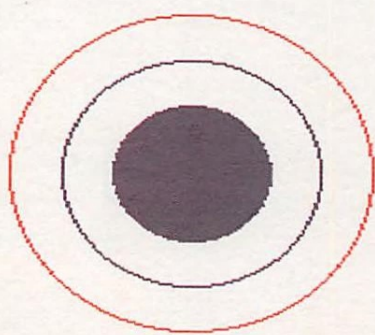
PAINT（X座標、Y座標）、塗る色［、境界色］

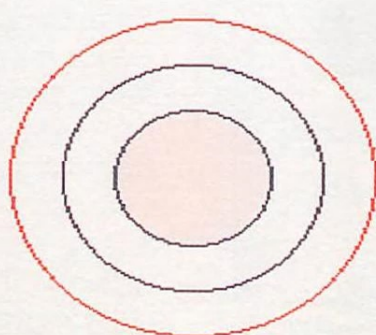
●X座標、Y座標	ここで塗りはじめる位置をグラフィック座標で指定する。STEPを付けて相対座標で指定することもできる。
●塗る色	文字どおり、ワクのなかを何色で塗るか指定する。SCREEN5の場合、指定色はカラーコードとなる。
●境界色	ワクの色を指定する。省略すると塗る色と同じ色が境界色になる。境界色は1色しか指定できないので注意。

■CIRCLE命令を使った元絵のサンプルリスト

```
10 SCREEN 5:COLOR 1,15,15:CLS
20 CIRCLE (128,96),30, 4
30 CIRCLE (128,96),50, 1
40 CIRCLE (128,96),70, 8
100 IF STRIG(0)=0 THEN 100
```

↑サンプルリストをRUNするとこんな画面になる。CIRCLEは円を描く命令だ

■PAINT命令の用例

```
50 PAINT (128,96), 4, 4
```

↑上のリストを追加すると、いちばん内側の青い円の中が塗られる。境界色を省略しても、結果は同じだ

```
50 PAINT (128,96),10, 4
```

↑上のリストのように、塗る色を10に変更すると、青い円の中を黄色で塗る。このとき、青い円のワクは画面に残る

```
50 PAINT (128,96), 4, 8
```

↑塗る色を青、境界色は赤に変えると、外側の赤い円の中を青く塗る。内側にある2つの円も一緒に塗られてしまう

SB進路相談

【質問：4月号の『PEGUINGS』の中で使われているユーザー定義関数のFNVは、いったいどんな意味があるのですか？　それとSOUND文とエスケープシーケンスを紹介してください／げんそO　福岡・15歳】　ユーザー定義関数のFNV(n)は、ペギングの周辺のカラーコードを調べて、8より大きければ－1、そうでなければ0を値とする。行10の2行目にある「POINT(X, Y－H)＞8」というのがこの関数の意味。

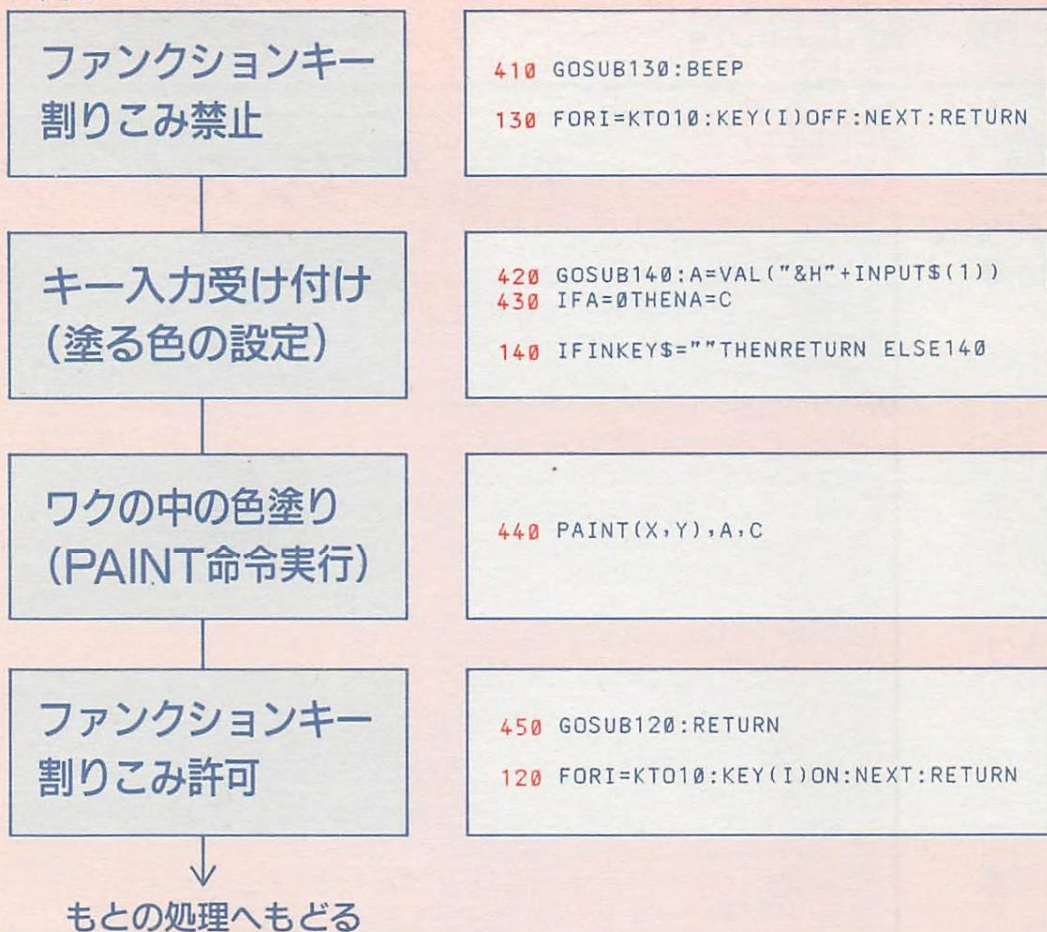
簡易線画ツールにPAINT処理を組みこむ

■PAINT処理の内容

●割りこみ処理の流れ

ファンクションキー割りこみ禁止

```
410 GOSUB130:BEEP

130 FORI=KTO10:KEY(I)OFF:NEXT:RETURN
```

ファンクションキーを使った割りこみ処理では、KEY(n)OFF命令を使って割りこみを禁止しておかないと、連続して割りこみがかかってしまうことがある。そこで、行130にある割りこみ禁止サブを呼び出して、まず割りこみを禁止する。つぎに、割りこみ処理が始まる合図として、ビープ音を鳴らしている。

キー入力受け付け（塗る色の設定）

```
420 GOSUB140:A=VAL("&H"+INPUT$(1))
430 IFA=0THENA=C

140 IFINKEY$=""THENRETURN ELSE140
```

PAINT命令では、塗る色と境界色の2つの色を指定するので、どちらか1つは割りこみ処理の中で設定する必要がある。使い勝手からマウスでワクを描いて、その中を何色で塗るか指定する方法にした。キー入力を受け付けるときは、先行入力された文字を消去しなければ誤動作の原因になるので、キーバッファクリアサブを新設した。

ワクの中の色塗り（PAINT命令実行）

```
440 PAINT(X,Y),A,C
```

塗る色の設定が終わったら、ワクの中をPAINT命令で塗るだけだ。塗り始めの座標はマウスカーソルの位置にし、境界色は、現在のカラーコードにした。あとは、それらの情報を、PAINT命令の書式に合わせて設定し、実行するだけだ。

ファンクションキー割りこみ許可

```
450 GOSUB120:RETURN

120 FORI=KTO10:KEY(I)ON:NEXT:RETURN
```

割りこみ処理を終えてもとの処理にもどるまえに、はじめのほうで禁止しておいた割りこみを、もういちど許可する必要がある。ここでは行120にある割りこみ許可サブを呼び出してから、RETURNでもとの処理にもどっているのだ。割りこみ許可サブにあるKEY(n)ONという命令は、ファンクションキーの割りこみを許可するものだ。

↓

もとの処理へもどる

では、先月掲載した『簡易線画ツール』に、PAINT処理を組みこんでみよう。

PAINT処理はファンクションキーの割りこみサブとして組みこむことにした（上参照）。

処理の中で行っていることは、大きくわけて4つある。まず、割りこみが重複しないように、割りこみ禁止にする。そして、PAINT命令のための入力を受け付ける。次にPAINT命令を実行し、最後に割りこみを許可してもとにもどるのだ。

割りこみ処理を組み終わったら、次に割りこみを定義しなくてはいけない。割りこみの定義は行30で行っているので、PAINT処理のある行番号を追加登録する。

右にあるリストがPAINT処理を組みこんだ完成リストだ。

完成リストでは、今後、処理を追加しやすいように、少し手を加えてあるが、大きな変更点はPAINT処理に関する部分と、新たに追加した、キーバッファクリアサブの2つだ。

まだまだツールにはほど遠い機能だが、少しずつ命令を紹介しながら機能を増やしていこうと思う。来月はLINE命令だ。

■操作方法の説明

操作方法は、F8キーでPAINTモードにしたあと、ビープ音が鳴ったら、塗りたい色を指定するだけだ。このとき、PAINT命令で指定されている境界色は、現在のカラーコードになっているので、いろいろな色でたくさんのワクを描いてから色を塗るときは、注意してほしい。PAINTするまえに、いったんセーブしておいたほうがいいだろう。

※付録ディスクには、以下のファイル名で収録してあります。
SAMPLE.SB6

⬆先月のバージョンで絵を描いてみた。線のみで描かれた単純な絵だ

⬆左の絵を今回のPAINT命令を使って着色してみた。

■簡易線画ツール6月号バージョン

```
10 SCREEN5,0:COLOR15,0,0:CLS
20 SPRITE$(0)="みたタ♠":C=15
30 K=8:ON KEY GOSUB,,,,,,,400,200,300:GO
SUB120
40 A=PAD(12):XX=X:YY=Y
50 X=X+PAD(13):IFX<0THENX=0 ELSEIFX>255T
HENX=255
60 Y=Y+PAD(14):IFY<0THENY=0 ELSEIFY>211T
HENY=211
70 A=VAL("&H"+INKEY$):IFA>0THENC=A
80 PUTSPRITE 0,(X,Y),C,0
90 IFSTRIG(1)THENLINE(XX,YY)-(X,Y),C
100 IFSTRIG(3)THENCLS
110 GOTO40
120 FORI=KTO10:KEY(I)ON:NEXT:RETURN
130 FORI=KTO10:KEY(I)OFF:NEXT:RETURN
140 IFINKEY$=""THENRETURN ELSE140
200 ' F9 キー サンプル:ガメン save ──────
210 GOSUB130:BEEP
220 BSAVE"SCREEN5 .GRP",0,&H69FF,S
230 GOSUB120:RETURN
300 'F10 キー サンプル:ガメン load ─────
310 GOSUB130:BEEP
320 BLOAD"SCREEN5 .GRP",S
330 GOSUB120:RETURN
400 ' F8 キー サンプル:PAINT サブ ─────
410 GOSUB130:BEEP
420 GOSUB140:A=VAL("&H"+INPUT$(1))
430 IFA=0THENA=C
440 PAINT(X,Y),A,C
450 GOSUB120:RETURN
```

SB進路相談

変数XとYはペギングの座標、HはFNV(n)のnにあたる（引数という）。ようするに、FNV(n)の中身はPOINT命令と関係演算で、スプライトとグラフィックの衝突判定をするための関数なのだ。ユーザー定義関数の意味を知りたいときは、定義している部分を見てみよう。次のエスケープシーケンスの紹介については、次号の進路相談でやってみようと思う。SOUND命令については、ちょっとむずかしい。このスペースでは紹介しきれないだろう。

超初心者 Super Beginners' 講座（スーパービギナーズ講座）

第19回 グラフィック入門その7

MSXFAN 付録ディスク →P118の「スーパー付録ディスクの使い方」参照

マウスを使ったほとんどすべてのツールには、直線を引いたり、円や四角を描く機能がある。そりゃそうだ。どんなに自由な線が描けるといっても、マウスで直線や円を描くのはかなり大変なのだから。

マウスでは描きにくい線

ほとんどのCGツールでは、マウスにより絵が描けるようになっていて、そのために自由に線が描ける利点がある。

しかし、いくらマウスで自由に線が描けるといっても、マウスにも苦手なものがあるのだ。

それは、前に紹介したようなプログラム的な線だ。直線や四角形、円といった、きちっとした図形をマウスで描くのはかなり苦しい作業になる。まあ、こういった図形を直接で描くのは、マウスに限らず難しい作業なのだが。

そこで、ほとんどのツールに直線や四角、円などの図形を描くためのコマンドやモードが用意されていて、それを使うことでこういった図形も扱えるようにしているのだ。

ところで、こういった図形は、どんな命令を使えば描くことができるのだろうか。

■LINEとCIRCLE

これらの図形(直線や四角形、円)を描くには、LINE命令とCIRCLE命令を使えばかんたんだ。

LINE命令を使うと、直線、長方形、中を塗りつぶした長方形の3つの図形を描くことができ、論理演算子も使える。

また、CIRCLE命令では、中心点の座標と半径のドット数、開始角、終了角、比率の指定によりさまざまな円を描くことができるのだ。

どちらの命令も、あとで紹介しているので読んでほしい。

なぜSCREEN5なのだろう?

このあいだ、ちえ熱が「どうしてSCREEN5なんだろう?」といっていた。どうやらちえ熱は、SB講座でSCREEN5を使っているのを見て、ファンダムやAVフォーラムの投稿作品に、SCREEN5を使っているものが多いことを思い出して、疑問を持ったようだ。

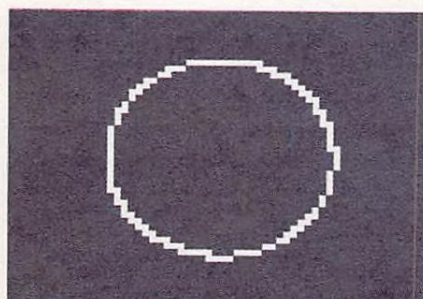
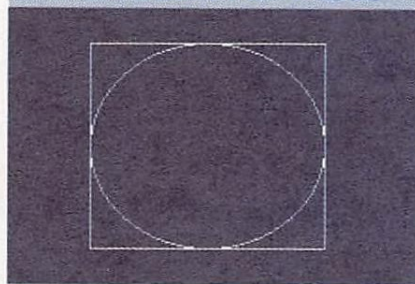
ところで、5月号のSB講座でグラフィック画面の各モードについて紹介していたのを覚えているだろうか?

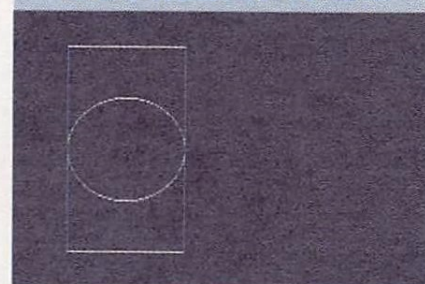
ここではもう一度、各画面モードの特徴と、なぜSCREEN5を使うのかを紹介しよう。

■2種類の画面モード

グラフィック画面には、大きくわけて2種類のモードがある。そのひとつがSCREEN2〜4の画面モードで、SCREEN2と4では色の使い方に制約があり、SCREEN3はドットがやたら大きく(左下の写真)使いづらい。

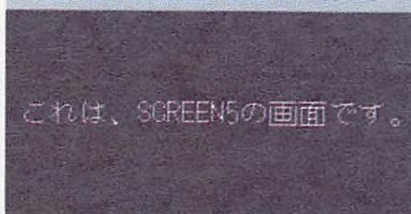
もうひとつがSCREEN5以降の画面で、COPYやSETPAGEなどの命令が使えるが、なかには横のドット数が倍になり細かいぶん見にくかったり、色数が少ない、または色数は多いがパレットがない画面モードがある。

これらの画面モードのなかで、SCREEN5の画面が、色数、ドット数とも標準的で扱いやすく、使われる頻度が高いのだ。

↑SCREEN3の画面に円を描くと、位置によってはいびつな円になる

SCREEN5の場合

SCREEN7の場合

↑下のリストはSCREEN5の画面に四角形と円を表示するもの。これをSCREEN7に変えて実行すると、四角形は縦長になるが、円はほぼ、おなじように表示される

```
10 SCREEN5
20 X=128:Y=96
30 CIRCLE(X,Y),64
40 LINE(X-64,Y-64)-(X+64,Y+64),15,B
50 GOTO50
```

SCREEN5の場合

これは、SCREEN5の画面です。

SCREEN7の場合

↑SCREEN5とSCREEN7の画面に文字(漢字)を表示させてみた。漢字に限らず、SCREEN5と7では横のドット数が違うので、文字の表示も細長くなるのだ

【エスケープシーケンスの紹介】 6月号のSB進路相談で、エスケープシーケンスを教えてほしいという質問というか、要望があったので、この欄外にて紹介しよう。まず、エスケープシーケンスというのはどんなものかを説明しよう。エスケープシーケンスは、CHR$(27)のエスケープコードというコントロールキャラクタと、ある特定の文字を組み合わせて、PRINT文で表示する文字列に指定したときに行える画面制御のことをいう。例えば、文

LINE命令の使い方

ＬＩＮＥ命令は、マウスでは描きにくい直線や四角形(長方形)を描くための命令なのだが、このＬＩＮＥ命令にはいろいろな使い方があるのだ。

もちろん、直線や四角形を描くことには変わりはないのだが、直線と四角形というだけでも2つの使い方があり、ほかにも四角形の中を塗りつぶしたり、論理演算子を付けて色を変化させたりできるのだ。

ＬＩＮＥ命令の書式は下にあるように、開始点と終了点の2つの座標、表示する色、機能、論理演算子と5つの情報を指定するようになっている。

また、5つの情報のうち、2つの座標以外は指定を省略することもできる。つまり、最低でも2つの座標さえ指定すれば、ＬＩＮＥ命令が使えるのだ。

■機能などの指定

まず、座標以外の指定について説明しよう。

ＬＩＮＥ命令でどんな図形が描かれるかは、機能の指定で決められる。

この機能の指定を省略すると直線を描き、"B"を指定すると四角形を描く。また、"BF"と指定すると四角形の中を塗りつぶした図形を描くのだ。

表示する色はそのままの意味で、省略するとCOLOR命令で指定された前景色が使われる。

論理演算子を付けると、表示される色と元の色とで論理演算が行われ、その結果の色が画面に残る。これを利用すれば、ある範囲の図形の色を反転することなどもできるのだ。

■座標の指定

ＬＩＮＥ命令は、開始点と終了点の2つの座標を指定して描くようになっている。

この2つの座標の指定方法にはいくつかのやり方があって、多くの場合、右のリスト1のように、画面の左上を基準としたグラフィック座標系で指定する絶対座標指定を用いる。

この他に、リスト2のようにＬＰ(最終参照点)を利用した指定方法がある。

ＬＰとは、最後に画面に表示(または移動)した点という意味で、グラフィック画面のカーソル位置のようなものだ。

開始点の指定を省略すると、ＬＰが開始点として扱われる。また、開始点を相対座標で指定したときは、ＬＰからの相対座標となる。

終了点を相対座標で指定したときは、開始点からの相対座標になる。終了点は省略することができないので注意が必要だ。

LINE命令の実行例

下のリストを実行すると、写真のように直線、四角形、中を塗りつぶした四角形の3つを表示する。

リスト1では、それらの図形を表示している3つのＬＩＮＥ命令には、絶対座標を使っている。

リスト2はリスト1の絶対座標指定を違う指定方法にしたものだ。

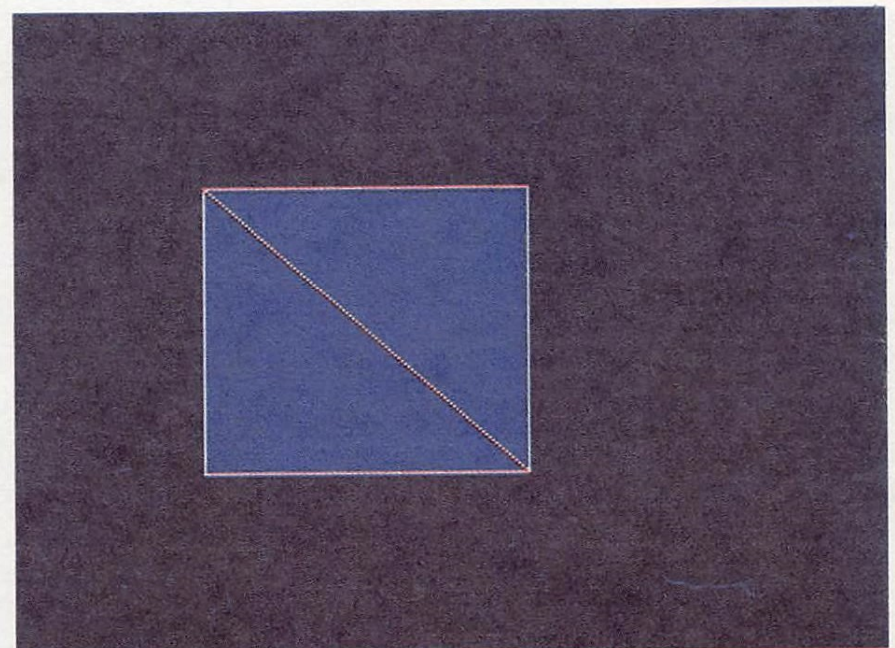
行20では終了点を開始点からの相対座標で指定しているし、行30では開始点の座標を省略。行40は開始点を省略したうえに、終了点を相対座標で指定しているのだ。

※打ちこんで試してみよう。

■リスト1

```
10 COLOR15,0,0:SCREEN5
20 LINE(50,50)-(150,150),5,BF
30 LINE(50,50)-(150,150),15,B
40 LINE(50,50)-(150,150),8
50 GOTO50
```

■リスト2

```
10 COLOR15,0,0:SCREEN5
20 LINE(50,50)-STEP(100,100),5,BF
30 LINE-(50,50),15,B
40 LINE-STEP(100,100),8
50 GOTO50
```

↑実行後は上の写真のような画面になる。行20のＬＩＮＥ命令では塗りつぶした四角形(青の四角)、行30では四角形(白の四角)、行40では直線(赤の線)を表示する

LINE(開始点の座標)ー(終了点の座標),カラーコード,機能,論理演算子

●開始点と終了点の座標指定

①絶対座標での指定	例：LINE	(XX, YY)ー(X, Y)
②相対座標での指定	例：LINE	STEP(＋x, ＋y)ー(X, Y)
	LINE	(XX, YY)ーSTEP(＋x, ＋y)
③開始点の省略	例：LINE	ー(X, Y)

【開始点の座標指定】 開始点の座標は絶対座標、ＬＰからの相対座標、ＬＰと3つの指定方法がある。このうち、ＬＰからの相対座標指定のときは、座標のまえにSTEPを付け、X座標、Y座標とも増分で指定する。また、開始点を省略すると、ＬＰを開始点として扱う。

【終了点の座標指定】 終了点の座標指定には、絶対座標と開始点からの相対座標指定のいずれかを用いる。終了点を相対座標で指定した場合、開始点のときとは違い、ＬＰからの増分ではないので注意が必要だ。また、終了点の座標は省略することができない。

【ＬＩＮＥ実行後のＬＰ】 ＬＩＮＥ命令を実行したあとは、ＬＰは終了点の位置に移動する。

●機能の指定

省略……2点を結ぶ直線の表示

B………2点を対角とする長方形を表示

BF……2点を対角とする長方形を塗りつぶして表示

●論理演算子の指定(SCREEN5以降で可能)

PSET(または省略)……指定色でそのまま表示

PRESET ………………指定色のNOTの色で表示

AND ……………………指定色と元の色とのANDの色で表示

OR………………………指定色と元の色のORの色で表示

XOR ……………………指定色と元の色とのXORの色で表示

SB進路相談 字の表示位置を指定するＬＯＣＡＴＥ命令とおなじ働きをするエスケープシーケンスとか、１行分画面をスクロールさせたり、１行新しい行を挿入したりできるのだ。変わったところで、カーソルの形を変えたり、見えなくしたりもできる。ただしこれらエスケープシーケンスは、ＰＲＩＮＴ命令でしか使用できないので、ファンクションキーに登録したりしても意味がないので注意しよう。まあ、ＰＲＩＮＴ命令と１セットと覚えていればいい。

CIRCLE命令の使い方

直線さえ描くことができれば、それを組み合わせてある程度の図形は描くことができる。しかし、こと円となると難しい。

そういった円を描く命令がCIRCLE命令なのだ。

CIRCLE命令は、真円を描くためだけの命令ではなく、比率を設定しての楕円や、開始角・終了角を設定して部分円を描くことだってできる。

CIRCLE命令の書式は、下にあるように円の中心の座標、半径のドット数、表示色、開始角、終了角、比率の6つの情報を指定することができる。

このうち、円の表示位置は中心点の座標で、円の大きさは半径のドット数で決まり、これだけ知っていればとりあえず円を表示することはできる。

表示色は円を描くときの色で、CIRCLEには論理演算子はないので、そのままの色で表示される。

開始角と終了角の指定は難しい。一口にいえば部分円を描くための円の描き始めの位置と、描き終わりの位置を角度によって指定するのだ。

比率は円のつぶれ具合を、縦方向と横方向の比率で指定するもので、この値を操作すると楕円を描くことができるのだ。

■CIRCLEの角度

さて、CIRCLEの開始角と終了角の説明をしよう。

右のリスト3は、開始角と終了角を指定したCIRCLEの例で、角度にはラジアンという単位を使用する。

時計でいえば、3時の位置から反時計回りに1周すると2πラジアンという値になるので、開始角がπラジアン、終了角が0(または2π)ラジアンなら9時の位置から反時計回りに3時の位置まで円を描くことになる。

角度の範囲は−2πから2πまでで、負のときは半径も描かれる。また、負の値でも、角度は絶対値の値になるので、−πもπも、おなじ位置となる。

ただし、πといってもグラフィックキャラクタのπではなく、数値のπ(3.14159……)のπなので注意しよう。

■比率

比率は楕円を描くためのもので、右のリスト4はその例だ。

比率の指定は、横方向に対して縦方向をどれくらいにするかの割合を指定するので、比率が1より大きければ縦長の楕円に、1より小さければ横長の楕円になる。例えば比率が2のとき、縦方向は横方向の2倍なので縦長の楕円になるというわけだ。

CIRCLE命令の実行例

リスト3は開始角と終了角を設定した場合の円の描かれ方の例。

行20はπの値を関数により計算し、P#という変数に設定(覚えておくと便利だろう)。

行30は開始角を0、終了角をπにして半円を描いている。

行40は開始角と終了角を入れ換えて反対側に半円を表示する。

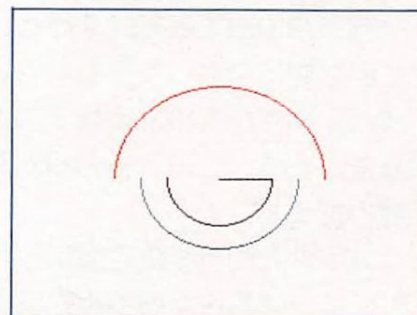
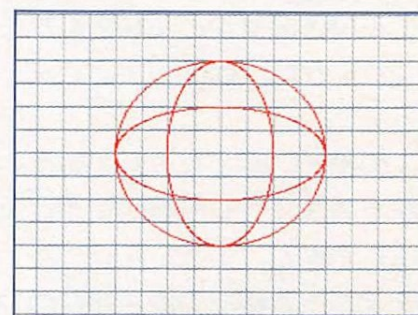
行50は終了角を−2πにして、終了角の位置から半径を表示している。0も2πも−2πもおなじ位置になることに注意しよう。

リスト4は比率を設定したときの例で、行20〜50のLINE命令でわかりやすいようにマス目を表示している。比率が1より大きいと縦長の楕円、1より小さいと横長の楕円になる。ただし、どのような楕円も、比率が1の半径がおなじ円より大きくなることはないので注意しよう。

■リスト3

```
10 COLOR1,15,15:SCREEN5
20 P#=ATN(1)*4
30 CIRCLE(128,96),64,8,  0, P#
40 CIRCLE(128,96),48,5, P#,  0
50 CIRCLE(128,96),32,1, P#,-P#*2
60 GOTO60
```

■リスト4

```
10 COLOR1,15,15:SCREEN5
20 FORI=0TO15
30 LINE(0,I*16)-(255,I*16),7
40 LINE(I*16,0)-(I*16,211),7
50 NEXT
60 CIRCLE(128,96),64,8
70 CIRCLESTEP(0,0),64,8,,,2
80 CIRCLESTEP(0,0),64,8,,,1/2
90 GOTO90
```

↑リスト3の実行画面。角度を指定すると部分円を表示できる

↑リスト4の実行画面。縦と横の比率がきれいに1対2になっている

■CIRCLE命令の書式

CIRCLE(円の中心点の座標),半径,カラーコード,開始角,終了角,比率

●円の中心の座標指定

①絶対座標での指定　例:CIRCLE (XX, YY)

②相対座標での指定　例:CIRCLE STEP(+x,+y)

●半径の指定

そのものズバリ半径のドット数を指定する。しかし、比率が1以外(楕円)のときを考えると、中心点からの最大距離という意味で理解しておいたほうが無難だ

●開始角、終了角の指定

−2πから2πまでの数値で、円を描き始める位置と、描き終える位置を指定する。角度は真右の位置から、反時計回りで2πで1周する。また、マイナスの数値を指定すると、中心点からその位置まで線が描かれる。ただし、マイナスの数値の場合、実際の角度には絶対値を使用しているので、−πとπ、−1/2πと1/2πなどのように、絶対値が同じものは同じ角度とみなされるので注意が必要。ちなみに、πの値は「ATN(1)*4」でかんたんにえられるので覚えておこう

●比率の指定

比率は楕円を描くための指定で、「Y方向の距離/X方向の距離」になっているので、1より小さければ横長の楕円になり、1より大きければ縦長の楕円になる。比率の指定を省略した場合、SCREEN 6と7に限り2(X方向の距離が2倍)になるので注意が必要だ

※CIRCLE命令を実行した場合、LPは中心点の座標になります。

【エスケープシーケンス一覧】 PRINT命令と組み合わせて使用する

●CHR$(27)+"Y"+CHR$(32+y)+CHR$(32+x)……カーソルをLOCATE x, yの位置に移動する

●CHR$(27)+"A"……カーソルを1文字ぶん上に移動する

●CHR$(27)+"B"……カーソルを1文字ぶん下に移動する

簡易線画ツールへの組みこみ

では、今回紹介したＬＩＮＥ命令とＣＩＲＣＬＥ命令を、先月の簡易線画ツールに組みこんでみよう。

下にある少し長めのリストがＬＩＮＥ命令とＣＩＲＣＬＥ命令を組みこんだ簡易線画ツール7月号バージョンのリストだ。

今回、ＬＩＮＥ命令とＣＩＲＣＬＥ命令を拡張するために、先月より多くリストを変更した。

変更した部分は、ＬＩＮＥ命令の処理とＣＩＲＣＬＥ命令の処理をファンクションキー割りこみに設定するために、行30を変更。さらに、ＬＩＮＥ命令とＣＩＲＣＬＥ命令の各処理を行500からと行600からに作成して追加した。

また、ＬＩＮＥ命令は2つの座標を、ＣＩＲＣＬＥ命令では中心点と半径の2つの情報が必要になり、使いやすさを考えると、マウスでそれらを設定できるようにしたほうがいい。

そこで、マウスによる座標計算の部分をサブルーチンにして、ＬＩＮＥ命令とＣＩＲＣＬＥ命令の各処理でも使用できるようにした。

このため、行40以降のメインルーチンとサブルーチン部分が変更されている。

簡易線画ツールの使い方は右の一覧をみてほしい。

各処理の詳しい解説は次ページからで行っているので、そちらもぜひ読んでほしい。

■F6、F7キーの割りこみ設定

```
30 K=8:ON KEY GOSUB,,,,,,,400,200,300:GOSUB100
```

```
30 K=6:ON KEY GOSUB,,,,,600,500,400,200,300:GOSUB100
```

■操作一覧

●メイン

【マウス】　カーソル移動
【右ボタン】　画面消去
【左ボタン】　線を描く

●ファンクションキー割りこみ処理

【Ｆ６キー】　カーソルの位置を中心とする円を描く
　マウス……カーソル移動（円の大きさ指定）
　右ボタン……キャンセル
　左ボタン……決定、円の表示

【Ｆ７キー】　カーソルの位置から直線、四角を描く
　①直線、四角の大きさ指定
　　マウス……カーソル移動（直線、四角の大きさ指定）
　　右ボタン……キャンセル
　　左ボタン……決定
　②直線か四角か入力
　　Ｌ(ｌ)……直線
　　Ｂ(ｂ)……四角
　　Ｆ(ｆ)……塗りつぶした四角

【Ｆ８キー】　現在の描画色で描かれたワクの中を塗る
　１～９、Ａ～Ｆ(ａ～ｆ)……塗る色の入力（16進コード）

【Ｆ９キー】　画面データの保存

【Ｆ10キー】　画面データの読みこみ

■簡易線画ツール（7月号バージョン）ファイル名:SAMPLE.SB7

```
10 SCREEN5,0:COLOR15,0,0:CLS
20 SPRITE$(0)="みたタ♠":C=15
30 K=6:ON KEY GOSUB,,,,,600,500,400,200,
300:GOSUB100
40 XX=X:YY=Y:GOSUB130
50 A=VAL("&H"+INKEY$):IFA>0THENC=A
60 PUTSPRITE 0,(X,Y),C,0
70 IFSTRIG(1)THENLINE(XX,YY)-(X,Y),C
80 IFSTRIG(3)THENCLS
90 GOTO40
100 FORI=KTO10:KEY(I)ON:NEXT:RETURN
110 FORI=KTO10:KEY(I)OFF:NEXT:RETURN
120 IFINKEY$=""THENRETURNELSE120
130 A=PAD(12)
140 X=X+PAD(13):IFX<0THENX=0 ELSEIFX>255
THENX=255
150 Y=Y+PAD(14):IFY<0THENY=0 ELSEIFY>211
THENY=211
160 RETURN
200 ' F9 ｷｰ ｻﾝﾌﾟﾙ:ｶﾞﾒﾝ save ──────────
210 GOSUB110:BEEP
220 BSAVE"SCREEN5 .GRP",0,&H69FF,S
230 GOSUB100:RETURN
300 'F10 ｷｰ ｻﾝﾌﾟﾙ:ｶﾞﾒﾝ load ──────────
310 GOSUB110:BEEP
320 BLOAD"SCREEN5 .GRP",S
330 GOSUB100:RETURN
400 ' F8 ｷｰ ｻﾝﾌﾟﾙ:PAINT ｻﾌﾞ ──────────
410 GOSUB110:BEEP
420 GOSUB120:A=VAL("&H"+INPUT$(1))
430 IFA=0THENA=C
440 PAINT(X,Y),A,C
450 GOSUB100:RETURN
500 ' F7 ｷｰ ｻﾝﾌﾟﾙ:LINE ｻﾌﾞ ──────────
510 GOSUB110:BEEP:XX=X:YY=Y
520 GOSUB130:PUTSPRITE 1,(X,Y),C,0
530 FORI=0TO1:LINE(XX,YY)-(X,Y),15,,XOR:
LINE(XX,YY)-(X,Y),15,B,XOR:NEXT
540 IFSTRIG(3)THEN580
550 IFSTRIG(1)=0THEN520 ELSEGOSUB120
560 BEEP:A=INSTR("LlBbFf",INPUT$(1))
570 IFA=0THEN560 ELSEIFA<3THENLINE(XX,YY
)-(X,Y),C ELSEIFA<5THENLINE(XX,YY)-(X,Y)
,C,B ELSELINE(XX,YY)-(X,Y),C,BF
580 IFSTRIG(1)ORSTRIG(3)THEN580
590 PUTSPRITE 1,(0,217):GOSUB100:RETURN
600 ' F6 ｷｰ ｻﾝﾌﾟﾙ:CIRCLE ｻﾌﾞ ──────────
610 GOSUB110:BEEP:XX=X:YY=Y
620 GOSUB130:PUTSPRITE 1,(X,Y),C,0
630 IFSTRIG(3)THEN670
640 IFSTRIG(1)=0THEN620
650 R=SQR((X-XX)^2+(Y-YY)^2)
660 CIRCLE(XX,YY),R,C
670 IFSTRIG(1)ORSTRIG(3)THEN670
680 PUTSPRITE 1,(0,217):GOSUB100:RETURN
```

SB進路相談

●ＣＨＲ＄（２７）＋"Ｃ"……カーソルを１文字ぶん右に移動する
●ＣＨＲ＄（２７）＋"Ｄ"……カーソルを１文字ぶん左に移動する
●ＣＨＲ＄（２７）＋"Ｈ"……カーソルをホームポジション（ＬＯＣＡＴＥ０，０の位置）に移動する
●ＣＨＲ＄（２７）＋"ｊ"またはＣＨＲ＄（２７）＋"Ｅ"……画面を消去する

直線や四角を描く処理を作る

LINE処理の組みこみ

ここでは、ＬＩＮＥ処理を簡易線画ツールの処理として作成する方法について紹介しよう。

■ＬＩＮＥ処理の仕組み

ＬＩＮＥ命令を使うとき、最低限必要な情報がある。それは開始点と終了点の２つの座標と、どのような図形を表示するかを指定する機能に関する情報だ。

開始点の座標は、ファンクションキーの割りこみがかかったときのカーソルの座標を使用し、終了点の座標はＬＩＮＥ処理の中で選択・決定するようにした。

また機能は、キー入力により選択することにした。

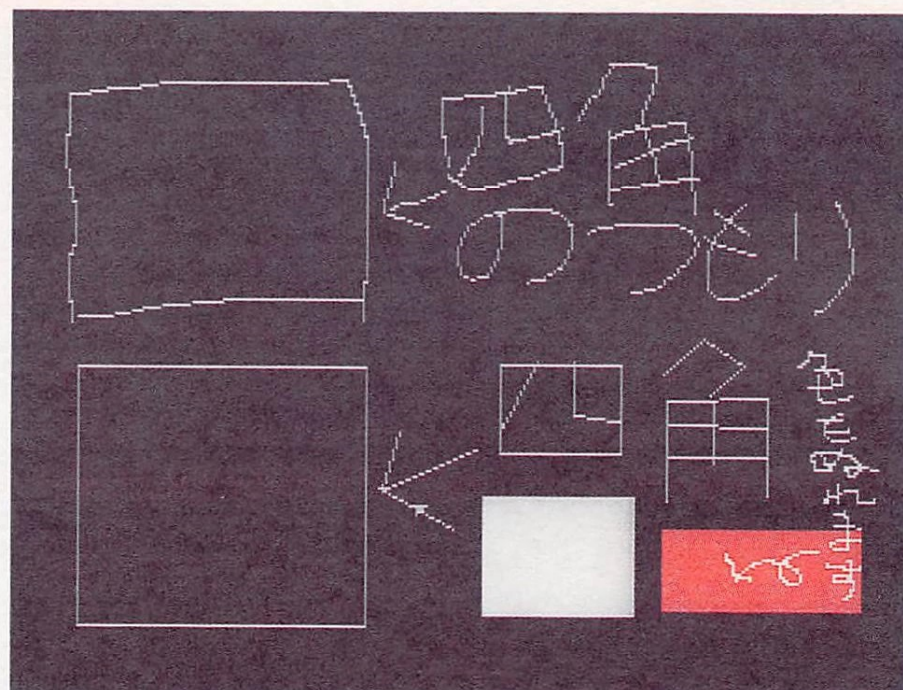
これで必要な情報が得られるのだが、いざ使ってみようとするとどのような図形が描かれるかわかりにくいという欠点がある。

そこで、ＬＩＮＥ処理のなかで、実際にどのように線が描かれるかをＬＩＮＥ命令を使って表示させることにした。

しかし、ＬＩＮＥ命令を使って実際に描いてしまうと、もともとあった絵が影響を受けてしまうことになるので、論理演算子のＸＯＲを活用することにした。

論理演算子のＸＯＲを活用すれば、ＬＩＮＥ命令で実際に図形を描いてしまっても、もういちどおなじ図形を、おなじ位置にＸＯＲを使って表示すると、もとの状態にもどってくれるのだ。

こうしてできたのが下で詳しく説明しているＬＩＮＥ処理だ。

これでマウスでは描きにくかった直線や四角が表示できるようになったので、試してみてほしい。

⬆直線や四角が描けるようになったので、上の写真のような絵でもかんたんに描けるのだ。マウスでこんな絵を描こうとしたら、きっとゆがんでしまうだろう

割りこみ禁止／開始点の座標保存

```
510 GOSUB110:BEEP:XX=X:YY=Y
```

●割りこみ禁止サブ

```
110 FORI=KTO10:KEY(I)OFF:NEXT:RETURN
```

ファンクションキー割りこみ処理で、まず最初にやることは、割りこみの禁止だ。ここでは、行110の割りこみ禁止サブを呼び出して割りこみを禁止している。また、処理を行うときに、カーソルの座標(開始点)を保存している。

終了点の選択、決定

・サブカーソル移動、表示
・ＬＩＮＥの例表示、消去
・決定入力受け付け（キーバッファクリア）

キャンセル入力受け付け

```
520 GOSUB130:PUTSPRITE 1,(X,Y),C,0
530 FORI=0TO1:LINE(XX,YY)-(X,Y),15,,XOR:
LINE(XX,YY)-(X,Y),15,B,XOR:NEXT
540 IFSTRIG(3)THEN580
550 IFSTRIG(1)=0THEN520 ELSEGOSUB120
```

●マウス入力サブ

```
130 A=PAD(12)
140 X=X+PAD(13):IFX<0THENX=0 ELSEIFX>255
THENX=255
150 Y=Y+PAD(14):IFY<0THENY=0 ELSEIFY>211
THENY=211
160 RETURN
```

●キーバッファクリアサブ

```
120 IFINKEY$=""THENRETURNELSE120
```

この部分がいちばんのポイントとなる部分で、ＬＩＮＥ命令を実行するときの終了点の座標を設定する処理だ。まず行130のマウス入力サブを呼び出して終了点の座標を更新し、サブカーソルのスプライトを表示する(行520)。

行530にあるＬＩＮＥ命令は、現在どのような図形が描けるかを試しに描く処理だ。ＬＩＮＥ命令の前後にあるＦＯＲ～ＮＥＸＴループで２回実行されるので、１回目にＸＯＲで線を引いても、２回目でもとにもどってしまうのだ。

行540はキャンセル判定で、右ボタンの入力があれば行580へ飛ぶ。行550で左ボタンの入力がなければ処理を繰り返し、入力があればキーバッファクリアサブを呼び出す。

ＬＩＮＥの種類入力

```
560 BEEP:A=INSTR("LlBbFf",INPUT$(1))
```

ＬＩＮＥ命令の機能の部分を入力する処理で、キー入力により選択される。

ＬＩＮＥ命令の種類は英字で行われ、Ｌなら直線、Ｂなら四角、Ｆなら塗りつぶした四角となる。ここではそれ以外の文字でも入力される。

ＬＩＮＥの実行

```
570 IFA=0THEN560 ELSEIFA<3THENLINE(XX,YY
)-(X,Y),C ELSEIFA<5THENLINE(XX,YY)-(X,Y)
,C,B ELSELINE(XX,YY)-(X,Y),C,BF
```

開始点と終了点、機能が決まったところで、ＬＩＮＥ命令を実行する部分。機能は変数では指定できないので、入力を判定して、それぞれ実行される。また、行560で入力がＬＢＦのどれでもなかった場合(ただし小文字は可)、もういちど行560を実行する。

入力終了待ち

```
580 IFSTRIG(1)ORSTRIG(3)THEN580
```

ここでは、マウスの左右のボタンの入力が、終了するまで待つ処理を行っている。

これは、ＬＩＮＥ処理を終えたときに線を引いたり画面を消してしまわないようにするために必ず必要になる処理だ。

スプライト消去／割りこみ許可

```
590 PUTSPRITE 1,(0,217):GOSUB100:RETURN
```

●割りこみ許可サブ

```
100 FORI=KTO10:KEY(I)ON:NEXT:RETURN
```

ＬＩＮＥ処理の中で使用した、サブカーソルのスプライトを消去する。そして、処理の始めで割りこみを禁止しているので、処理を終えるまえに、行100にある割りこみ許可サブを呼び出して、ファンクションキーの割りこみを許可しているのだ。

●ＣＨＲ＄(２７)+″Ｌ″……カーソルのある位置に１行挿入して以降の行を下へスクロールする
●ＣＨＲ＄(２７)+″Ｍ″……カーソルのある位置を１行削除して以降の行を上へスクロールする
●ＣＨＲ＄(２７)+″Ｋ″……カーソルの位置から右側にある文字を消去する

円を描く処理を作る

CIRCLE処理の組みこみ

簡易線画ツールに、こんどは円が描けるようにCIRCLE処理を組みこむやり方を紹介しよう。

ところで、CIRCLE処理を組むまえに、CIRCLE命令のどこまでを組み入れるかが問題になってくる。

この問題は、どこまでの情報を設定すればいいかにつながるので、あらかじめ決めておかなくてはいけないことなのだ。

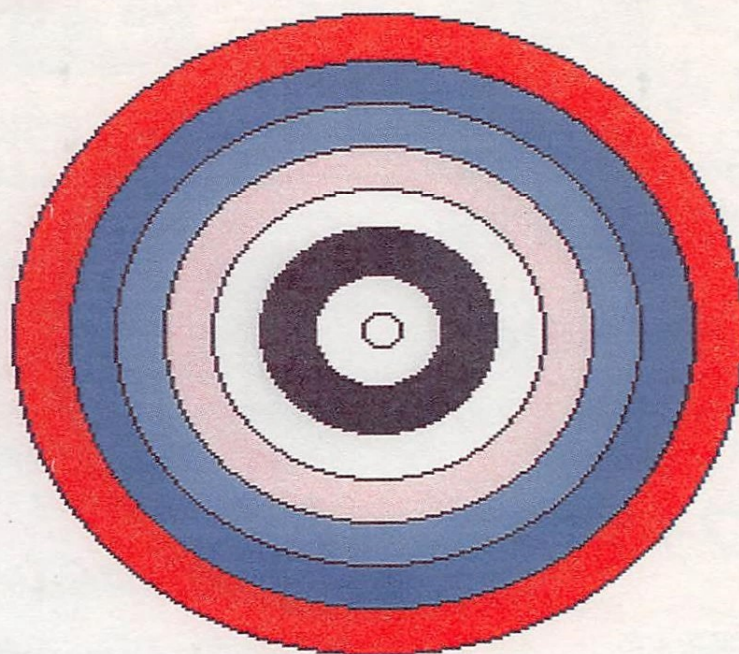
まず比率だが、これが指定できるようになれば楕円が描けるのでかなり便利だろう。しかし、比率をどのように指定したらいいのだろうか？ CIRCLE命令の場合、論理演算が使えないので、実際に描いて見せるわけにはいかないし、楕円を描くためには、最低でも通過点を2か所指定しなくてはいけない上、面倒な計算も必要になってくる。

開始点、終了点もおなじように最低でも2か所の指定を必要とするし、ある程度の計算式も必要だ。

そこで、簡易線画ツールなのだから、操作が面倒になるこれらの要素は切り捨てることにした。

結果的にこのCIRCLE処理で必要な情報は、中心点の座標と半径の2つになった。

中心点の座標は割りこみがかかったときのカーソル座標を使用し、半径は処理の中で円の通過点を指定して、かんたんな公式で計算することにした。

こうしてできたのが下で説明しているCIRCLE処理で、これで直線や四角に加えて、円も描くことができるようになったのだ。

⇧LINE処理に加えてCIRCLE処理が使えるようになったので、きっちりした絵が描けるようになった。さて次は、どんな機能を増やそうかな

割りこみ禁止 中心点の座標保存

```
610 GOSUB110:BEEP:XX=X:YY=Y
```

●割りこみ禁止サブ

```
110 FORI=KTO10:KEY(I)OFF:NEXT:RETURN
```

LINE処理とおなじように、ファンクションキーの割りこみを禁止する必要がある。ここでも行110の割りこみ禁止サブを呼び出す。

また、CIRCLE命令を実行するときに、中心点の座標で使用する、現在のカーソル座標の保存も行う。

通過点の選択、決定

・サブカーソル移動、表示

・決定入力受け付け

キャンセル入力受け付け

```
620 GOSUB130:PUTSPRITE 1,(X,Y),C,0
630 IFSTRIG(3)THEN670
640 IFSTRIG(1)=0THEN620
```

●マウス入力サブ

```
130 A=PAD(12)
140 X=X+PAD(13):IFX<0THENX=0 ELSEIFX>255
THENX=255
150 Y=Y+PAD(14):IFY<0THENY=0 ELSEIFY>211
THENY=211
160 RETURN
```

LINE命令とおなじように、CIRCLE処理でも半径をマウスによって設定する処理を行っている。まず、行130のマウス入力サブを呼び出し、円の通過点の座標を更新し、サブカーソルのスプライトを表示する(行620)。

CIRCLE命令には論理演算子が使えないので、LINE処理のように実際にどのような図形になるかを表示させられない。そこで、少しでも使いやすくするために、円がどこを通過するかで半径を指定することにしたのだ。

行630はキャンセル判定で、右ボタンの入力があれば行670に飛ぶ。

行640では左ボタンの入力がなければ行620からの処理を繰り返す。

半径の計算

```
650 R=SQR((X-XX)^2+(Y-YY)^2)
```

ここでは、通過点の座標と円の中心点の座標から、CIRCLE命令によって描かれる円の半径を計算している。この計算式についてここでは触れないが、中学校あたりの数学の授業で習う、かなり基礎的な公式だ。

CIRCLEの実行

```
660 CIRCLE(XX,YY),R,C
```

いままでの処理の中で得られた情報をもとに、CIRCLE命令を実行して円を表示する。

この処理では、開始角も終了角も比率も設定していないので、設定できるように工夫してみるといいだろう。

入力終了待ち

```
670 IFSTRIG(1)ORSTRIG(3)THEN670
```

ここでもLINE処理とおなじように、マウスの左ボタン、右ボタンの入力が終わるのを待っている。こういった、入力が継続してしまわないようにするための処理は必要なので、心掛けておいてほしい。

スプライト消去 割りこみ許可

```
680 PUTSPRITE 1,(0,217):GOSUB100:RETURN
```

●割りこみ許可サブ

```
100 FORI=KTO10:KEY(I)ON:NEXT:RETURN
```

CIRCLE処理の中で使用した、通過点を指定するサブカーソルのスプライトを消去する。

そして、処理の始めで禁止しておいたファンクションキーの割りこみを、行100の割りこみ許可サブを呼び出して許可する。最後にRETURNでもとの処理へもどる。

SB進路相談

●CHR$(27)+"J"……カーソルの位置から画面の終わりまで消去する
●CHR$(27)+"x4"……カーソルを「■」の形にする
●CHR$(27)+"x5"……カーソルを消す
●CHR$(27)+"y4"……カーソルを「＿」の形にする
●CHR$(27)+"y5"……カーソルを表示する

スーパー ビギナーズ Super Beginners' 講座

超初心者

第20回
グラフィック入門その8

付録ディスク →P116の「スーパー付録ディスクの使い方」参照

今月のSB講座には、簡易線画ツールのリストは掲載されていない。なぜなら、もっと使いやすいツールにしようと作り直したら、10画面ほどのリストになってしまったのだ。これはもう、「簡易」とは呼べない。

※SB講座への質問や紹介してほしい内容などを募集します。ハガキで、MSX・FAN編集部「SB進路相談」までお寄せください。

グラフィック画面の「色」

つい最近のことだが、7月号の校了(Mファンが印刷されるまえに、きちんと指定した色で印刷されているかなど、最終的なチェックを行う作業)のとき、ファンダム班の新人がSB講座に書かれていた「前景色」という言葉に疑問を持ったようで、チェックが入っていた。

ファンダムでは常用句になっているこの言葉も、MSX初心者にはわからないのは当然のことだろうし、初心者でなくとも知らなければわからないだろう。

またファンダムでは、カラーコードやパレットコードなど、「色」に関しての言葉が入り乱れているので、何かと混乱の種になっている場合が多い。

今月は、パレット切り換えを紹介しようと思っていたので、もう少し範囲を広げて、MSXの「色」について紹介しよう。

■カラーコードとパレットコードの違い

カラーコードもパレットコードもおなじだと思っている人は、案外多いのではないだろうか。

この2つの言葉は、言葉の意味を考えると違うことがわかる。

カラーコードは、LINE命令やCIRCLE命令などで、色を指定するときに使う数値で、これは画面モードによって範囲が変わるものだ。

例えば、SCREEN5なら0～15の数値がカラーコード、SCREEN8なら0～255の数値がカラーコードになる。

この範囲の数値には、それぞれ対応する「色」があって、多くの画面モードの場合、パレットコードとダブっている。

パレットコードというのは、MSX2以降の機種から使うことができるようになった、「色」が登録できる16個の箱の番号のことだ。またこの言葉は、登録されている「色」を直接示すことが多い。

このように、カラーコードとパレットコードでは意味が違うのだが、ほとんどの画面モードでは「カラーコード=パレットコード」となっているので、おなじだと思われていてもおかしくはないのだ。

しかしSCREEN6と8の画面モードでは、右表のようにカラーコードはパレットコードの一部とおなじか、全然違う内容になっているので注意が必要。

■カラーコードを調べる命令

カラーコード関連の命令に、指定座標のカラーコードを調べるPOINTという命令がある。

ここでは命令と紹介したが、POINTは数値関数なので、実際に使う場合は、

数値変数=POINT(X, Y)

などのようにしなければエラーが出てしまうので注意しよう。

■COLOR命令の書式

COLOR 前景色, 背景色, 周辺色

【前景色】 文字や点、線の画面への表示色でカラーコード(※)で指定する。省略することもできるが、省略した場合は色は変化しない

【背景色】 画面の背景の色で、カラーコード(※)で指定する。前景色と同様に省略することもできる。テキスト画面の場合は、命令実行とともに背景色も変化するが、グラフィック画面ではCLS命令などで画面が初期化されないと画面の色は変化しない

【周辺色】 画面のまわりの部分の色で、カラーコード(※)で指定する。周辺色も省略することができる

※指定するカラーコードの数値の範囲は、SCREEN 6の場合は0～3、SCREEN 8では0～255、その他はパレットコードの0～15

■SCREEN6の色

- ●使用できるカラーコードは0～3のパレット
- ●他の画面からSCREEN 6に変更した場合、画面の色は下位2ビット(4で割った余り)の色になる
- ●SCREEN 6で画面の色を設定した場合、他の画面モードに変更すると、0～3のカラーコードはそれぞれ0、5、10、15に変わる

例：COLOR15, 4, 7の状態でSCREEN 6にすると
COLOR15, 4, 7 ⇒ COLOR3, 0, 3
SCREEN 6、COLOR3, 1, 2で画面モードを変更すると
COLOR3, 1, 2 ⇒ COLOR15, 5, 10

■SCREEN8の色

- ●使用できるカラーコードは0～255でパレットはない
- ●カラーコードと表示色との対応は、

b7	b6	b5	b4	b3	b2	b1	b0
緑の成分			赤の成分			青の成分	

b7～b0はカラーコードの各ビット
※緑と赤は8段階、青のみ4段階

カラーコード=【緑の成分】×32+【赤の成分】×4+【青の成分】

例：カラーコード255(&B11111111)=白
　　カラーコード224(&B11100000)=緑

■POINT命令の書式

変数=POINT(X座標, Y座標)

【X座標、Y座標】 座標は絶対座標で指定。また、指定する数値は実際の座標上限値を超えるものや、マイナスの値も指定可能

【変数】 POINT命令によって指定された座標のカラーコードが入る。ただし、指定座標が画面外の場合は、−1が結果として得られる

※POINT命令は関数なので、変数への代入文やIF～THEN命令の条件式などで使用

SB進路相談

【質問：①SCREEN1～3でスプライトモード2は使えないのでしょうか？ ②VDP(n)という命令は、一体どういうものなのでしょうか？／加藤剛 埼玉・16歳】 まず、①の質問のSCREEN1～3の画面モードではスプライトモード1しか使えない。これらの画面モードは、MSX1との互換性を保つために受け継がれてきたものなので使えないのだ。しかし、特殊なテクニックを使うと、SCREEN1でスプライトモード2を使うことができる。これはVDP関

パレット切り換え

色関連のファンダム用語に、「パレット切り換え」というものがある。これは前述のパレットコードに登録された「色」を変更することを意味している。

ここではそのパレット切り換えについて紹介しよう。

■パレットの色

MSXの電源を入れて、

COLOR15，4

と実行して何か文字を入力してみよう。すると、文字の色つまり前景色が白になり、バックの色つまり背景色は青になる。

COLOR命令を使って前景色をカラーコード15、背景色をカラーコード4にしたのだから、カラーコード15は白、カラーコード4は青なのだとわかる。

正確に言えば、テキスト画面(プログラムが打ちこめる画面)では、カラーコード＝パレットコードなので、パレットコード15が白で、パレットコード4が青だということになる。

このパレットコードの「色」は、どのように決められているのだろうか。

■パレットの成分

画面の色は、赤、緑、青の3つの光の強弱で表現されている。これは光の3原色というもので、右上の図のように、組み合わせることで多くの「色」を表現することができるのだ。

MSXのパレットコードでは、赤、緑、青の光の度合いを個々に設定できるようになっていて、MSXの電源を入れた直後や、画面モードを変更したときには、右下の表のように各パレットコードが初期化される。

パレットコード15が白なのは、この初期化によって赤、緑、青の値が設定されるからなのだ。

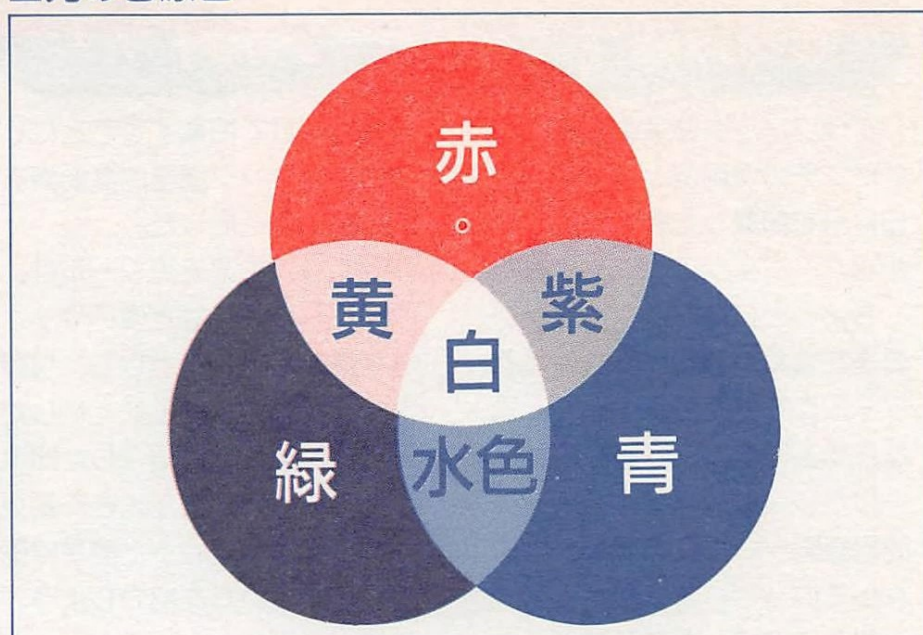
このパレットの赤、緑、青の値を変更する命令が、下にある「COLOR=」という命令で、このことをパレット切り換えと呼んでいるのだ。

赤、緑、青のそれぞれの値は、0～7の数値で指定するので、色の組み合わせは全部で、

8×8×8=512色

ということになる。

いろいろと試してみるといい。

■光の3原色

■初期状態のパレット

パレット番号	赤	緑	青
0：透明	0	0	0
1：黒	0	0	0
2：緑	1	6	1
3：明るい緑	3	7	3
4：青	1	1	7
5：明るい青	2	3	7
6：暗い赤	5	1	1
7：水色	2	6	7

パレット番号	赤	緑	青
8：赤	7	1	1
9：明るい赤	7	3	3
10：黄色	6	6	1
11：明るい黄	6	6	4
12：暗い緑	1	4	1
13：紫	6	2	5
14：灰色	5	5	5
15：白	7	7	7

■「COLOR=」命令の書式

COLOR=(パレットコード，赤の成分，緑の成分，青の成分)

【パレットコード】 切り換えを行うパレットコードを指定する。ただし、パレットコード0は透明色に固定されているので、パレット切り換えするときは、

VDP(9)=VDP(9) OR &H20

を実行しなければできない。これを実行後、透明色にもどすには、

VDP(9)=VDP(9) AND &HDF

を実行すればいい

【各色の成分】 赤、緑、青のそれぞれの成分を、0～7の8段階の数値で指定する。全部の成分が0なら黒、7なら白というように、数値が大きいほどその明るさが増す

COLOR=NEW

切り換えたパレットの状態を、初期状態の色にもどす

COLOR=RESTORE

VRAMに保存されているパレット情報を読みこみ、パレットを切り換える。これはおもに、パレット切り換えを行って描かれ、そのパレット情報ごと保存されたグラフィックを画面に読みこんだとき、この命令を実行してパレットを復帰するのに使用される

SB進路相談 数やBIOSなどといった部分を使用するのでここでは答えられないが、近いうちにスクラムで紹介しよう。SCREEN 2 をスプライトモード 2 にするとSCREEN 4 とおなじだ。②のVDP(n)というのは、MSXの中にあるVDPレジスタというものに、データを設定したり読み出したりするための関数だ。VDPレジスタはそれぞれ決まった役割を持ち、画面表示に関するほとんどのことに係わっているものなのだ。

簡易線画ツールのバージョンアップ

グラフィック命令の紹介とともに、その使用例ということで5月号で掲載した簡易線画ツールをバージョンアップしてきた。

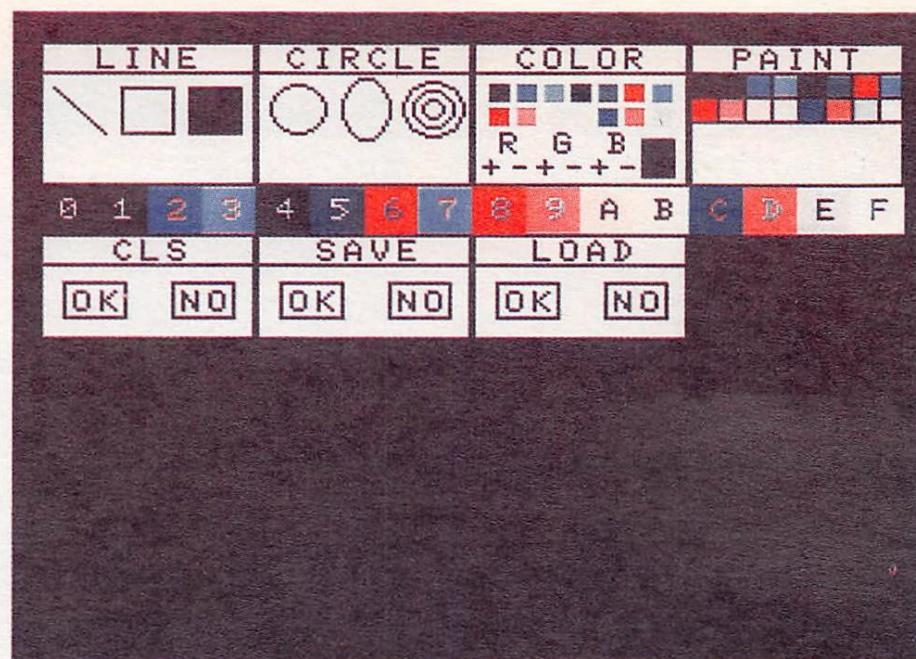
ちょっとしたプログラミング講座を兼ね、ある程度は使えるツールにできれば幸いと、二兎ならぬ三兎を追ってみたのだ。

しかし、機能が増えるに従い、簡易線画ツールは不便なものになっていってしまった。

そこで、扱ったグラフィック命令のプログラムでの使用例は、必要に応じて掲載することにし、プログラミング講座的な要素を取り除くことにした。

かわりに簡易線画ツールは、ウインドウ表示など使いやすさを考慮したツールとして、付録ディスクに収録することにした。

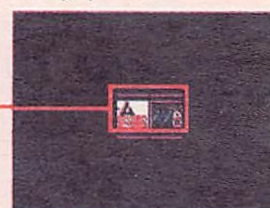
ここでは、今月紹介した命令が加わり、使い方が大きく変更され、生まれ変わった簡易線画ツールの使い方を紹介しよう。

※ファイル名
EDITOR−5. SB8

ページ2の画面にプログラムによって描かれるウインドウのグラフィック。実行時にはCOPY命令によって、神出鬼没に画面に現れる。開いている部分は未完成のモード用だ

■予定している機能

【F1】 直線、長方形の描画
【F2】 円、楕円の描画
【F3】 パレットの切り換え
【F4】 枠の中の色塗り
【F5】※グラフィック複写・移動
【F6】 画面消去
【F7】※文字の入力・表示
【F8】※ドット修正⇨部分拡大
【F9】☆グラフィックの保存
【F10】☆グラフィックの読みこみ

※予定のコマンド
☆機能が途中までのもの

■ウインドウ表示のしくみ

※ページ2のウインドウ保存画面は、複写元になるだけなので変化していないため、修復する必要はない。

F1 LINE

直線、長方形、塗りつぶした長方形を描画する

F1キーを押すとLINEモードになる。現在の表示色で直線、長方形などを描くことができる。

ウインドウが表示されたら、直線、長方形、塗りつぶした長方形のうち、描きたい図形にマウスでカーソルを移動して選択する。

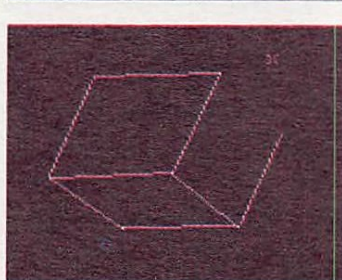
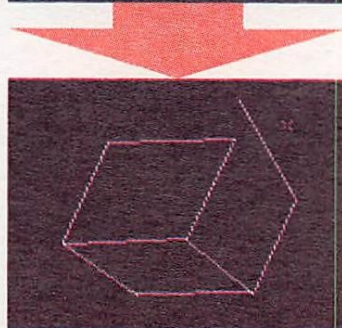
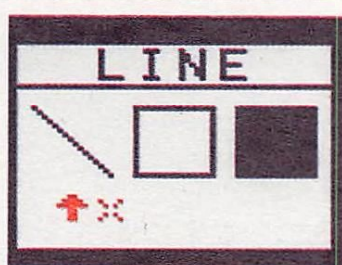
選択されている図形の下には、矢印カーソルが表示されるので、それでよければ左ボタンで決定し描画モードになる。

描画モードでは、まず開始点をマウスで指定し、左ボタンで決定。

次に終了点をマウスで指定し、左ボタンで決定する。

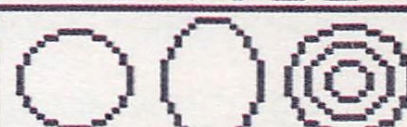
終了点を決定すると図形が描かれ、再び開始点の指定を行う。

右ボタンを押すと、1つまえの状態にもどり、ウインドウ表示時に押すとモードを終了する。

描画モードでESCキーを押せば、直前に描いた図形のみ、消すことができる。

描く図形を決定したら、いよいよ図形の描画だ。左ボタンをうまく使えば、連続した線も描画できるのだ

失敗したらESCキーを押せば描き直せる

F2 CIRCLE

円と楕円を描画する

F2キーでCIRCLEモードになり、現在の表示色で、真円、楕円、同心円が描ける。

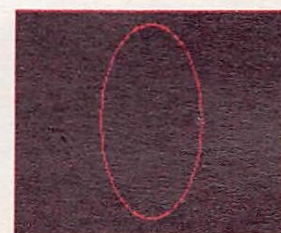
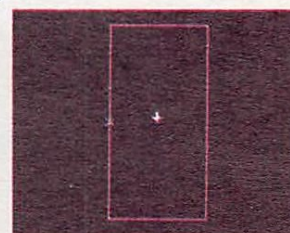
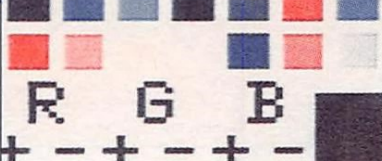
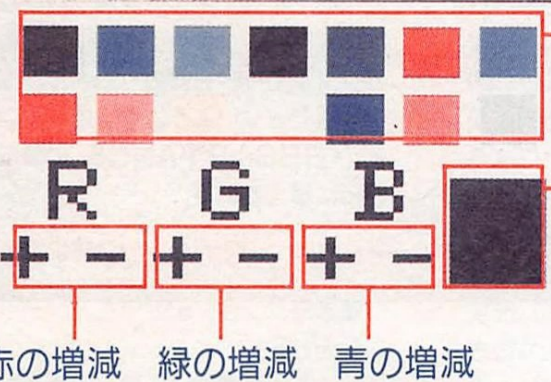
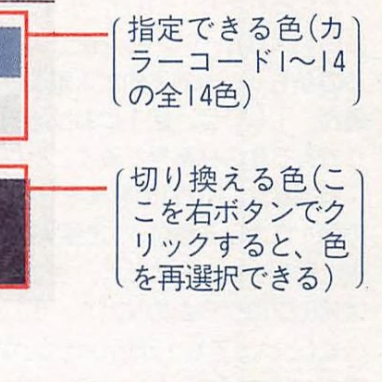
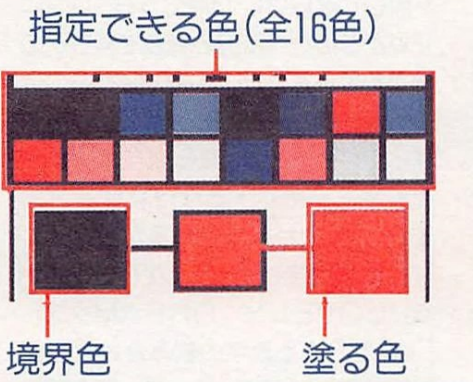
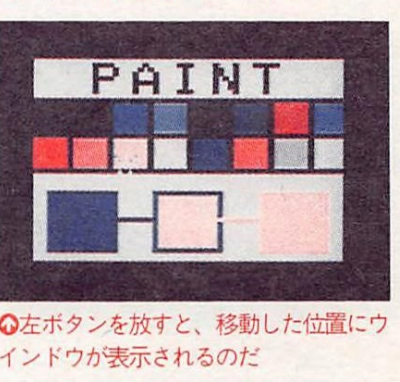
操作の基本はLINEモードとおなじ。ただし、描画モードではそれぞれの図形により異なる。

【真円】 中心点を指定し、次に円の通過点を指定する。描画後はまた中心点の指定になる

【楕円】 中心点を指定し、次に縦の大きさを指定、最後に横の大きさを指定すると描画される。描画後はまた中心点の指定になる

【同心円】 真円の場合とおなじように中心点と通過点を指定する。描画後は中心点が固定されるので、通過点の指定をするだけで円が描かれる。右ボタンで中心点の指定にもどる

同心円は中心点がおなじ円をいくつも描くのに、とても便利だ

■楕円の操作手順

楕円を描くときは、どんなサイズになるか長方形が表示されるのでわかりやすい

【質問:ターボRが最近どうも調子が悪いようで、①フォーマットしてもたまにFile not foundが出る ②ディスクの中に、「A:¥」という見覚えのないファイルが現れ、それを消すとすべてのデータが消える ③FILESをしても、Disk I/O errorとなってLOADできないことがある/柏崎芳史 岩手・17歳】 ①フォーマット中にFile not foundのエラーは出ない。フォーマットしたディスクにFILESなどをすれば必ず出る ②「A:¥」の「A:」はドライブA

F3 COLOR

パレットを切り換える

F３キーを押すとパレットコード１～14のパレットを切り換えることができる（０と15は不可）。
まず、切り換えたいパレットをマウスで選択・決定する。
パレットを決定したら、R、G、Bの文字の下にある「＋」や「－」にカーソルを合わせ、左ボタンを押すとその色の成分を増減できる。
また、切り換えている色の表示部分にカーソルを合わせ、左ボタンを押すと色を決定し、再度切り換える色の選択にもどる。

パレットコードーの色を切り換えているところ。絵を描いてからのほうが色を決めやすいぞ

指定できる色（カラーコード1～14の全14色）
切り換える色（ここを右ボタンでクリックすると、色を再選択できる）
赤の増減　緑の増減　青の増減

F4 PAINT

枠の中の色塗り

F4キーを押すとＰＡＩＮＴモードになる。
まず枠（境界色）の色にカーソルを合わせ左ボタンで決定。次に同様の操作で塗る色も決定する。
するとウインドウが消えるので、塗りたい部分にカーソルを移動し左ボタンを押すと色が塗られる。
色塗り時に右ボタンを押すと、境界色の選択からやり直す。
Ｆ１～Ｆ４の機能では、モード名を左ボタンで押すとウインドウの位置を変更できる（写真参照）。

この部分にカーソルを合わせて左ボタンを押すと……

ウインドウが消えるので、左ボタンを押したまま好きな位置に移動する

左ボタンを放すと、移動した位置にウインドウが表示されるのだ

指定できる色（全16色）
境界色　塗る色

F6 CLS

画面を消す

F６キーを押すと、画面を消去することができる。
画面にウインドウが表示されるので、よければ「ＯＫ」にカーソルを合わせ、表示が反転したら左ボタンで決定、画面を消去する。
間違って消してしまっても、すぐＥＳＣキーを押せば復活する。

黒く反転表示されたら選択していることになる

F9 SAVE

グラフィックをディスクに保存する

F９キーを押すと、作成したグラフィックと現在のパレットの状態をフロッピーディスクに保存することができる。
操作方法はＣＬＳとおなじで、保存するなら「ＯＫ」にカーソルを合わせ、左ボタンで実行する。
現在のバージョンでは、ＢＳＡＶＥ形式の全画面の保存となっているが、完成時にはグラフィックの部分保存やファイル名入力も可能にする予定でいる。
また、ディスク関係のエラー処理は一切していないので、せっかく描いたグラフィックもエラーが出ると消えてしまうので注意。

F10 LOAD

グラフィックをディスクから読みこむ

F10キーを押すと、あらかじめフロッピーディスクに保存しておいたグラフィックを読みこみ、パレットも設定する。
操作方法はＣＬＳやＳＡＶＥとおなじで、グラフィックを読みこむなら「ＯＫ」にカーソルを合わせ、左ボタンで実行する。
現在のバージョンではF９キーのＳＡＶＥに対応したものとなっているが、これもＳＡＶＥの機能アップに伴いグラフィックの部分読みこみやファイル名入力も可能にする予定でいる。
ＳＡＶＥ、ＬＯＡＤで扱えるグラフィックのファイル名は、
ＳＣＲＥＥＮ５　．ＧＲＰ
のみとなっている。

右ボタン 色変更

通常時に右ボタンを押すと、画面中段にカラーボードが表示され、表示色を変更することができる。
マウスで選択、左ボタンで決定、右ボタンなら表示色は変わらない。

矢印カーソルで使いたい色を選択して左ボタンで決定だ

ESC 色検出

通常時にＥＳＣキーを押すと、カーソル位置のカラーコードを検出し、表示色として取りこむ。
検出時にどんな色かを知らせる四角がカーソルの周りをまわる。

実際はカーソルの周りを四角がグルグルまわるのだ

SB進路相談
という意味で、「¥」はDOS2のルートディレクトリを表す記号だ。A高校の生徒をファイルに例えれば、「A高校」がルートディレクトリにあたる。A高校がなくなると生徒もいなくなる。だからファイルが全部消えるのだ。「A：¥」は気にしなくていい　③はっきりした原因はわからないが、おそらくディスクに問題があるはずなので、フォーマットしなおそう。だいじょうぶ、柏崎くんのターボRは正常だと思うよ。

スーパー Super ビギナーズ Beginners' 講座

超初心者

第21回
グラフィック入門その9

今月のSB講座では、グラフィック画面に文字を表示する方法を紹介しよう。テキスト画面に文字を表示するのとはかなり違うので、しっかり覚えておこう。

※SB講座への質問・リクエストを募集します。ハガキで、MSX・FAN編集部「SB進路相談」まで。

グラフィック画面に文字を表示する

今月ファンダムに掲載された1画面プログラムは、どの作品でもグラフィック画面を使っている。LINEやCIRCLEなどを使えば、かんたんに画面が作れるからだ。

今まで紹介してきた命令を使えば、採用作品のように画面を作るのはかんたんだ。しかし、スコアやタイトルなどの文字は、どうやってグラフィック画面に表示するのだろうか。

■グラフィック画面に文字を表示するには

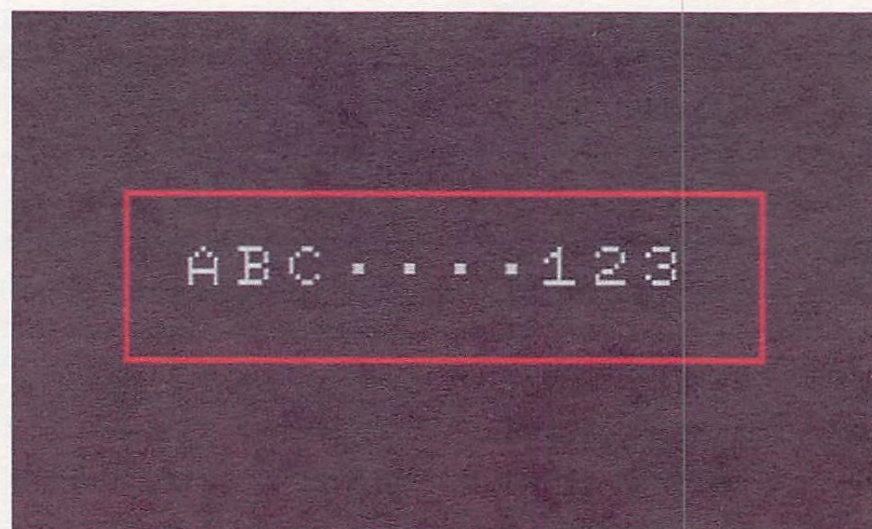
グラフィック画面では、テキスト画面のように、キーボードから入力した文字が表示されることもないし、LOCATEやPRINTを使っても画面には何も表示されない。

グラフィック画面に文字を表示するには、テキスト画面とはちょっと違うやり方をしないといけないのだ。

その方法とは、下のカコミでも紹介しているが、
OPEN"GRP:"～
という命令を実行し、MSXに"グラフィック画面に文字を表示させるための準備"をさせる必要があるのだ。そして、実際に文字を表示する命令が、
PRINT#～
という命令で、テキスト画面でのPRINTとほとんどおなじ働きをしてくれる。

キーボードにふれてみよう①

下のリストを打ちこんで実行すると、画面に文字が表示される(写真参照)。行10を見てもわかるように、これはグラフィック画面に書かれた文字なのだ。行20でグラフィック画面に文字を表示するための準備をし、行30で実際に表示するのだ。PSETについては次ページを読もう。

```
10 SCREEN5:COLOR15,0,0:CLS
20 OPEN"GRP:"FOROUTPUTAS#1
30 PSET(100,100),1:PRINT#1,"ABC····123"
40 LINE(90,90)-(190,120),8,B
50 GOTO50
```

グラフィック画面だとわかりやすいように赤い四角も表示される

■グラフィック画面に文字を表示するのに使う命令

●あらかじめ使用する命令

OPEN"GRP:" FOR OUTPUT AS#1

もしくは、

OPEN"GRP:" AS#1

グラフィック画面をファイルとして開き、出力可能な状態にする。グラフィック画面に文字を表示するためには、あらかじめ、このどちらかを実行しておかなければいけない。単純に、グラフィック画面に文字を表示するために必要なおまじないと思っていればいい。また、この命令は、RUN、CLOSE、ENDなどの命令を実行すると無効になる。

●実際の表示命令

PRINT #1,

PRINT #1, USING～

シーケンシャルファイルに対する出力命令。ここでは、テキスト画面のPRINT命令やPRINTUSING命令とおなじ働きと思っていればいい。表示位置に関しては、次のページを参照すること。

SB進路相談

【自作のCGツールを作ったのですが、それはSCREEN5で、SCREENを7にして描いたグラフィックをセーブすると、半分から上までしかセーブできません。どうにかして下の部分も出す方法はないのですか？　また、部分的に画面に出すことはできないのですか？　VRAMに関係あると思うのですが、アドレスは&H69FFにしてます／上田義？　広島・15歳】　上田くんの推測はズバリ的中しているのだ。たぶん、自作したSCREEN5用の

表示後の座標と論理演算

OPEN"GRP:"〜はPRINT#をテキスト画面のPRINTのように使うために必要なもので、これさえ実行しておけば、文字を表示することができる。しかし、テキスト画面にあるLOCATEのような働きをするものは、グラフィック画面にはないのだろうか。

■表示位置の指定

グラフィック画面に文字を表示する場合、表示される位置はLPによって決まる。前回紹介したLINEやCIRCLEでもLPが登場したが、LPとは、グラフィック画面でのカーソル位置のようなもので、例えば、LINE(Ø, Ø)-(1Ø, Ø)と実行したなら、LPの位置は(10, 0)に移動する。

つまり文字の表示位置を指定するには、直前でグラフィック命令を実行してLPを移動させればいいのだ。

グラフィック命令といっても、LINEなどを使うと余計な線を描かれたりして困る。そこで、PSET命令を使うといいだろう。PSETを使えば、LPも指定できるし、論理演算の恩恵にも授かれるので便利だ。

■グラフィック画面に表示される文字と論理演算の関係

右にあるのは、PSETを使って文字を表示するとき、論理演算を行ったときの例だ。

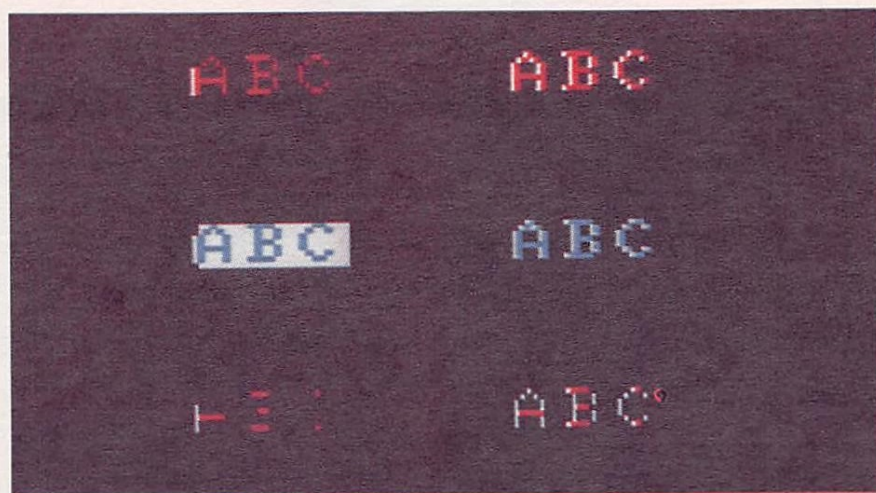
グラフィック命令で論理演算が行われると、MSXはその論理演算を覚えているのだ。そのために、グラフィック画面に文字を表示すると、その論理演算の影響を受けて表示が変化するのだ。これを利用すれば、重ね合わせて太文字にしたり、色を反転したりできて楽しいので、いろいろ試してみよう。

!!キーボードにふれてみよう②

これはPSETで位置指定をすると同時に論理演算を指定したときの例。

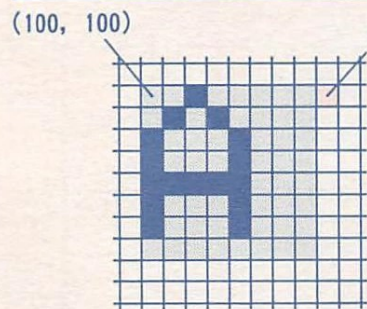
まず「ABC」という文字を6つ表示し(行40)、その文字の上に、文字の色をカラーコード8(赤)に変えて、論理演算を指定して1ドットずらして重なるように表示(行50〜100)してみたものだ。

どれも白い「ABC」を表示しただけのものだが、「ABC」の上に赤い「ABC」を表示しただけのものだが、論理演算によって影響を受けていることが、はっきりわかるだろう。

```
10 SCREEN5:COLOR15,0,0:CLS
20 OPEN"GRP:"FOROUTPUTAS#1
30 M$="ABC"
40 FORI=0TO2:FORJ=0TO1:PSET(90+J*50,I*30
+50),0:PRINT#1,M$:NEXTJ,I:COLOR8
50 PSET(91,50),0,PSET:PRINT#1,M$
60 PSET(141,50),0,TPSET:PRINT#1,M$
70 PSET(91,80),0,PRESET:PRINT#1,M$
80 PSET(141,80),0,TPRESET:PRINT#1,M$
90 PSET(91,110),0,AND:PRINT#1,M$
100 PSET(141,110),0,TAND:PRINT#1,M$
110 GOTO110
```

↑上からPSET、PRESET、ANDとなっていて、右側の列はTを付けて透明色は演算の対象にしないようにしている。文字の色や論理演算を変えて試してみよう

■文字を表示したあとのLP

```
10 COLOR15,4,7:SCREEN5:OPEN"GRP:"AS#1
20 PSET(100,100),4:PRINT#1,"A";
30 COLOR9:PRINT#1,"「ここが LPです。"
40 GOTO40
```

※以下、行10、30、40は共通。

●「;」がある場合

上のリストの行20のように、表示する文字や数値の最後に「;」(セミコロン)を付けた場合、表示後のLPは図のように最後に表示された文字の右上より1ドット右にずれた位置に移動する。

ただし、移動後のLPの座標が画面外のときは下を参照。

```
20 PSET(100,100),4:PRINT#1,"A"
```


●「;」がない場合

表示するメッセージの最後に「;」がないときは、LPは8ドット下のいちばん左端に移動する。

例えば、行20が上のような場合、LPは左の図の位置に移動するのだ。

「;」が付かなかった場合のLPの位置は、最後の文字が画面の右端になり、「;」がある場合とおなじ位置に移動するので、左下のところも参考にしよう。

```
20 PSET(248,100),4:PRINT#1,"A";
```

●最後の文字が画面の右端になり、「;」がある場合

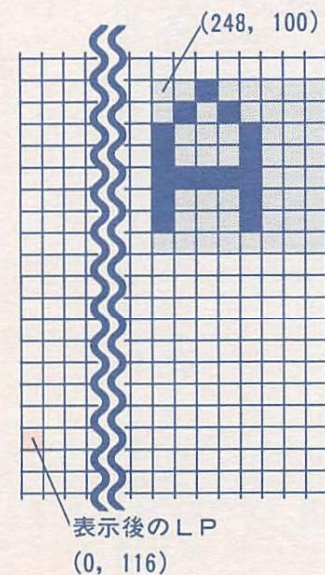
表示するメッセージの最後の文字が画面の右端になった場合、「;」があってもLPは8ドット下の左端に移動する。

例えば、行20を上のように変更すると、LPは左の図の位置に移動する。

テキスト画面でのカーソルの動きとおなじなので、覚えやすいだろう。

ただし、移動後のY座標が画面外のときは、LPは画面左上(0, 0)の位置に移動するので注意しよう。

また、画面の右端に文字を表示するとき、X座標が248(SCREEN6や7なら504)を超えると、文字が変形してしまうので、これにも注意してほしい。

```
20 PSET(248,100),4:PRINT#1,"A"
```

●「;」がなく、最後の文字が画面の右端になった場合

表示するメッセージの最後に「;」がなく、さらにメッセージの最後の文字が画面の右端になったとき、LPは16ドット下の左端に移動する。

例えば、行20が上のようになっているときは、LPは左の図の位置に移動するのだ。

実際に変更して実行してみると、"ここがLPです。"の文字が、"A"の位置とはずいぶん離れたところに表示されるのがわかるだろう。

また、この場合も右端の文字のX座標が248(または504)より大きくならないように注意が必要だし、移動後のY座標が画面外のときは、画面左上(0, 0)の位置にLPが移動するのでこれにも注意しよう。

SB進路相談

CGツールを、SCREEN7に変更して絵を描いていたのだろう。結論からいえば、アドレスを&HD3FF(パレット情報を含めるなら&HFA9F)に変更すればいいのだ。ただし、半分しか保存されなかったグラフィックはそのままなので、潔くあきらめたほうがいい。もうひとつの「部分的に画面に出す」というのは、COPY命令を使えば実現できる。COPY命令については、次回からのSB講座で紹介するのでそれまで待ってほしい。

スーパー Super ビギナーズ Beginners' 講座

超初心者

第22回
グラフィック入門その10

グラフィック入門もいよいよ大詰め。MSX2以降から加わった、COPY命令を紹介する。この命令ひとつでグラフィック画面をじゅうぶん活用できるハズだ。

※SB講座への質問・リクエストを募集します。ハガキで、MSX-FAN編集部「SB進路相談」まで。

COPY命令とは

SCREEN5以降のグラフィック画面での最大の魅力は、今回紹介するCOPY命令だと断言できるほど、この命令には魅力があるのだ。

■COPY命令とは

COPYには「複写する」とか「写す」といった意味がある。コピーといえば、まず思いつくのがコピー機だろう。いまでは家庭用なんかも普及しているし、コピー機が設置されているコンビニも多いので、夜中でも利用できてとても便利だ。

COPY命令にはそういった意味にふさわしい役割があって、あるときはグラフィックを複写したり、またあるときはディスクに保存されたファイルを複写したりする働きを持っている。

今月紹介するのは、このようにたくさんあるCOPY命令の使い方の中から、グラフィックをほかの場所に複写する方法だ。

■VRAM内でのCOPY

グラフィックをほかの場所に複写するという目的だけでも、画面のほかの部分への複写はもちろんのこと、ディスクや配列変数にも複写できるのだ。

また、論理演算子を指定することで、グラフィックを重ね合わせることもできるし、長方形でしか複写できないが、大きさも自由なので、アニメーションなどで威力を発揮する。

しかし、どこからどこへ複写するかによってCOPY命令の書式が決まるので、今月は画面から画面へ、つまりVRAM内でのCOPYを紹介する。

VRAM内でのCOPY命令の書式は右にまとめておいた。

基本的には、転送元の範囲をLINE命令のように長方形の対角の座標で指定し、転送先の指定は、複写を開始する位置を座標で示すことで行う。

ただし、SCREEN5以降の画面ではページという概念があるので、ほかのグラフィック命令とは違い、どのページからどのページへ複写するかという指定も必要になってくる。

ちょっと面倒だし、ページがからむとミスも起こりやすい。しかし、逆にいえば、ページを切り換えることなくVRAMを自由に扱えるので魅力だ。

せっかくページの話をしたのだから、SETPAGE命令についてもかるく紹介しておこう。

SETPAGE命令は、画面に表示されるページ(ディスプレイページ)と、線を描いたりするページ(アクティブページ)を決める命令で、書式は、SETPAGE〈ディスプレイページ〉〈,アクティブページ〉となっている。また、SCREEN命令を実行したあとは、どちらもページ0になっている。

■VRAM内でのCOPY命令の書式

COPY【転送元領域】TO【転送先】【,論理演算子】

【転送元領域】で示されたVRAMの領域を、【転送先】で示された位置に複写する。
おもに、おなじ図形をたくさん使う場合や、【論理演算子】を指定しての図形の重ね合わせに使用する。
このCOPY命令は、SCREEN5以上の画面モードのみ、使用可能。

例、ページ1の左上から縦横32ドットぶんをページ0の(100,100)の位置に複写する。

```
     COPY(Ø, Ø)-(31, 31), 1  TO  (1ØØ, 1ØØ), Ø
または、COPY(Ø, Ø)-STEP(31, 31), 1  TO  (1ØØ, 1ØØ), Ø
```

●【転送元領域】の指定方法
・(転送先開始点)-(転送元終了点),図形のあるページ
・(転送元開始点)-STEP(終了点までの増分),図形のあるページ

転送元開始点の座標は、絶対座標で指定しなければいけない。また、省略はできない。
転送元終了点の座標は、絶対座標でも相対座標でも指定できる。これもまた、省略はできない。
図形のあるページは、SCREEN5と6は0~3、その他の画面では0または1。また、SETPAGE命令で設定されているアクティブページとおなじ場合には省略できる。

●【転送先】の指定方法

・(転送先開始点),転送先のページ

転送先の座標は、絶対座標で指定しなければいけない。また、省略することもできない。
転送先のページは、SCREEN5と6は0~3、その他の画面では0または1。また、SETPAGE命令で設定されているアクティブページとおなじ場合には省略できる。

●【論理演算子】の指定方法
論理演算子には、以下の10種類が指定できる。論理演算子を省略することもできるが、その場合はPSETを指定したのとおなじ結果になる。

・PSET……………図形をそのまま複写。省略時とおなじ
・PRESET………図形の色を反転して複写
・OR…………………図形と転送先の状態で1ドットごとにORをとる
・XOR………………図形と転送先の状態で1ドットごとにXORをとる
・AND………………図形と転送先の状態で1ドットごとにANDをとる
・TPSET…………図形の透明色以外の部分をそのまま複写
・TPRESET……図形の透明色以外の部分を反転して複写
・TOR………………図形の透明色以外の部分と転送先の状態で1ドットごとにORをとる
・TXOR……………図形の透明色以外の部分と転送先の状態で1ドットごとにXORをとる
・TAND……………図形の透明色以外の部分と転送先の状態で1ドットごとにANDをとる

SB進路相談

【よくSCREEN1のゲームで、字の形や色を変えて使っているけど、SCREEN5や7、8で、文字を変えて表示するにはどうしたらいいのですか。特にSCREEN5の場合を教えてください。ただし、論理演算を使って、影を付けたりするのではなくて、形自体を変えたいのです。／清水雅浩 福岡・13歳】 グラフィック画面でキャラクタパターン定義をしたいという質問は、ずいぶん昔からよく送られてくるのだが、どうやって答えたらいいものか。

COPY命令を使う

ここでは、COPY命令を使うときに指定する座標についての注意をしたいと思う。

■転送元領域の指定での注意

グラフィックの転送元の指定では、転送元開始点と転送元終了点の2つの座標を指定するが、おなじ大きさのグラフィックを複写するのにも、2つの座標を指定する方法はいくつかある。

たとえば、右の写真の場合、複写するグラフィックの左上の座標を転送元開始点、右下の座標を転送元終了点に指定したために、転送元のグラフィックも転送先のグラフィックも壊れてしまっている。まあ、この場合、転送先の座標が、転送元のグラフィックに重ならない場所にあれば問題ないのだが。

しかし、もしこのとき、転送元開始点と転送元終了点を入れ換えて、転送先の座標を、転送が終了する位置に変えたらどうだろうか。こうすれば、転送元のグラフィックは壊れてしまうが、転送先のグラフィックはきれいに複写されるのだ。

COPY命令では、いっぺんに複写されているように見えるけど、じつは1ドットずつ順番に複写されているので、どこから転送するかをよく考えて指定するようにしよう。

■転送先の指定での注意

前の例では、転送先の座標の指定にも問題があるように思えるが、ときにはわざと転送先の座標を転送元のグラフィックに重ねることもある。それについてはページを改めて紹介するとして、意外と盲点になりやすい転送先の指定での注意をしよう。

転送先の座標を指定するときに注意してほしいのが、複写が完了したときに、どこまでグラフィックが転送されるかだ。

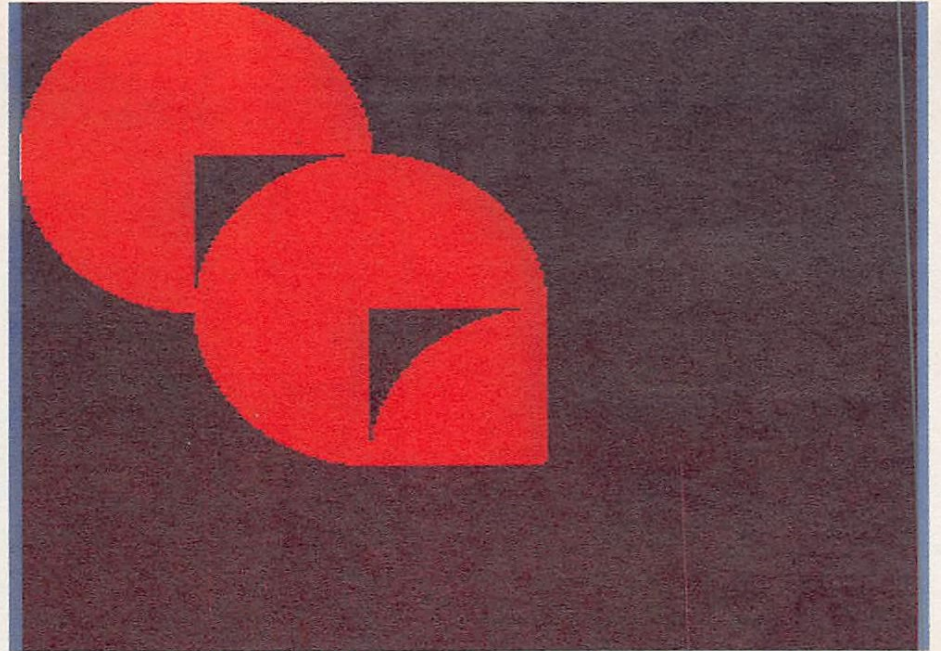
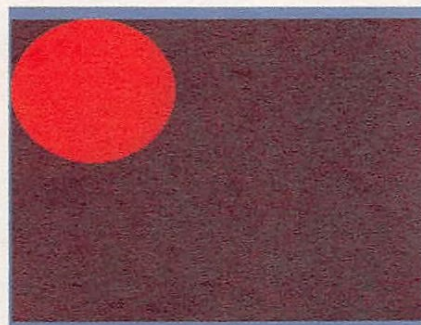
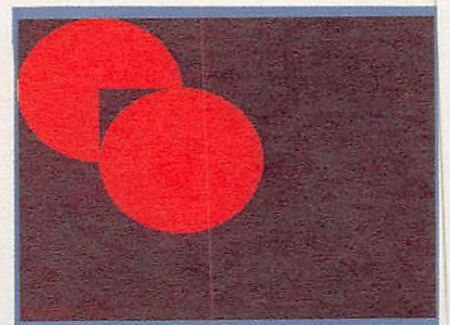
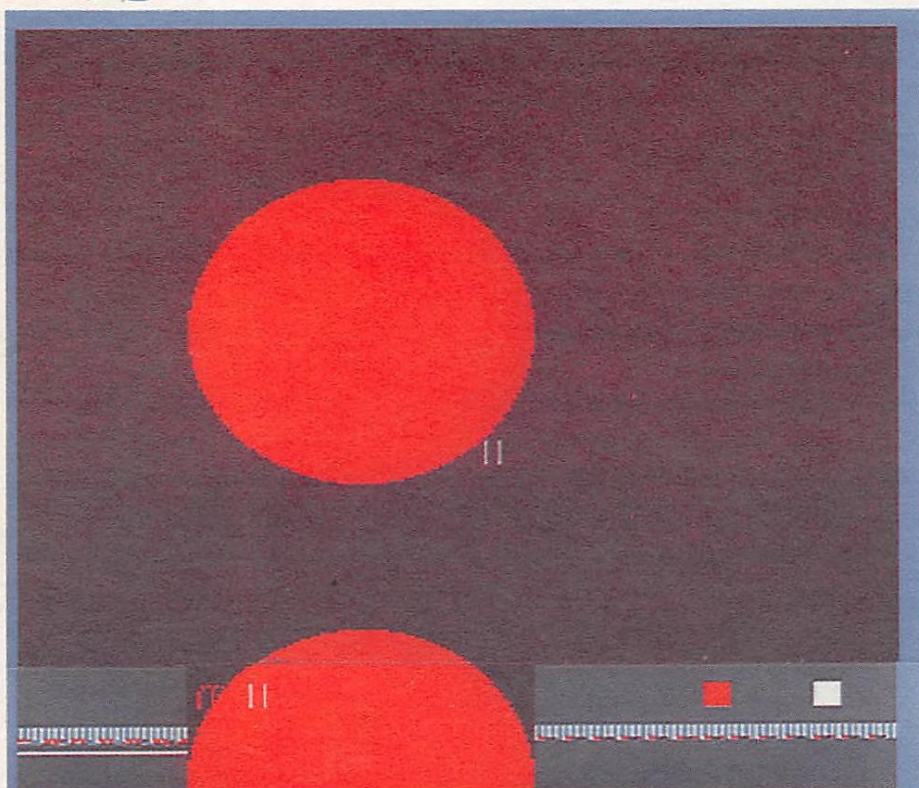
下の写真を見てほしい。これは極端な例だが、先ほどとおなじ赤い丸のグラフィックを、画面の下のほうに複写したときのページ0とページ1の状態だ。

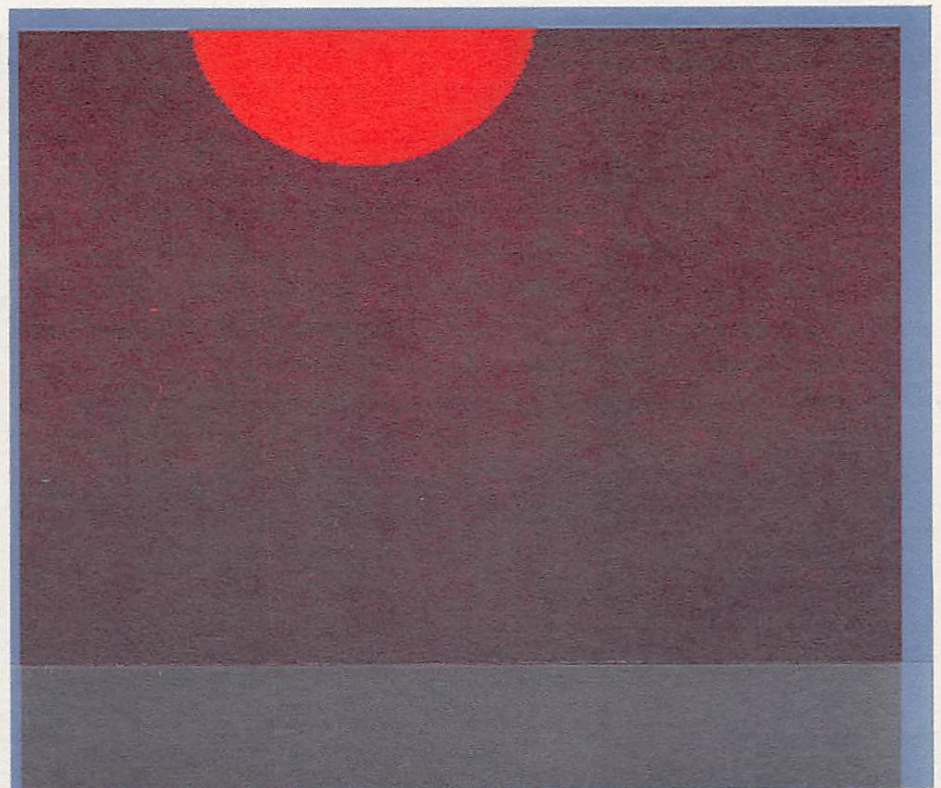
実際の画面では、写真の下のほうの部分は表示されないが、ここにはパレットやスプライト関係のデータが保存されている。つまり、この部分は絵を描くところではないわけだ。

ところが、赤い丸を複写したために、このデータ領域が破壊されて、画面のところどころに妙なスプライトが表示されてしまっている。そして、それでも入りきらなかったグラフィックの一部が、次のページにまではみ出してしまっている。

VRAMでは、絵を描く部分以外、つまり、画面に表示されない部分には、このようなデータがあることが多いので、こういうことが起こってしまうのだ。

ほんとうに極端な例なのだが、過去ファンダムに投稿された作品のなかには、こういった状態になってしまうものもあった。

COPY命令では、転送先の開始点しか指定できないので、転送が終了したときに、グラフィックがどこに複写されるかを注意してほしい。

↑複写は転送元開始点から順に行われる。だから、転送している途中で、まだ転送していない部分が変わってしまうと、その状態で転送されてしまうのだ

↑これが複写前の状態。赤い丸が複写するグラフィックだ

↑開始点と終了点を入れ換えれば、転送先のグラフィックはちゃんと複写される

ページ0

↑実際の画面では、写真の下のほうの色が違う部分は表示されない。この部分はパレットやスプライト関係のデータが保存されているので、画面には変なスプライトが表示されてしまう

ページ1

↑ページ0に入りきらなかった部分が、次のページ1にはみ出してしまった。画面の各ページはつながっているので、こういうことも起こるのだ。でも、なんかおもしろい

SCREEN1では、VRAMに文字の形を保存しているところがあるので、そこを書き換えてやればかんたんにできる。しかし、グラフィック画面では文字の形をVRAMには保存していないのだ。では、どうして文字が表示されるかというと、文字の基本的な形の情報がROMに記録されているので、そこからデータを持ってきているのだ。ほら、もう難しくなってきたでしょ。結論をいえば、グラフィック画面でもキャラクタパターン定義はできる。⇨

COPY命令の応用

グラフィック命令で指定する論理演算子というと、LINE命令や、文字を表示するときに利用することぐらいしか思いつかない。なにしろ、論理演算子は変数などで指定するわけにはいかないので、場合によっての使い分けが面倒で、色コードを始めから計算して指定したほうがよっぽどいいからだ。

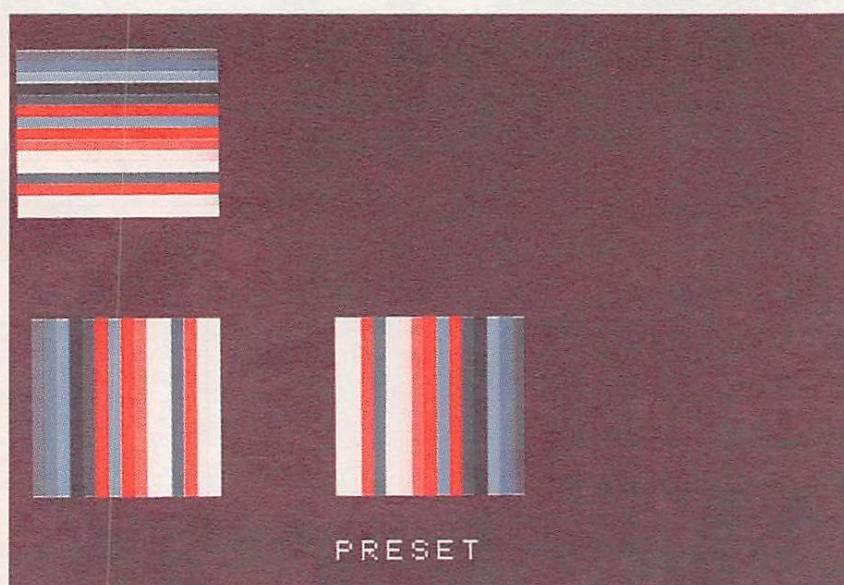
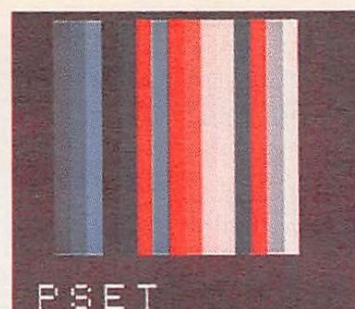
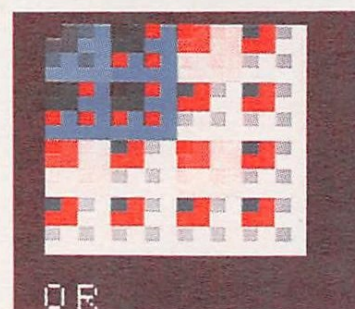
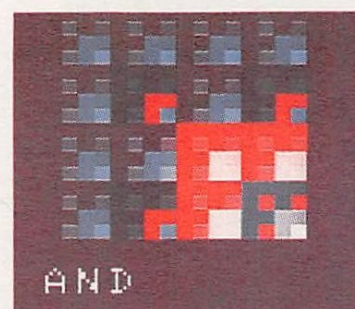
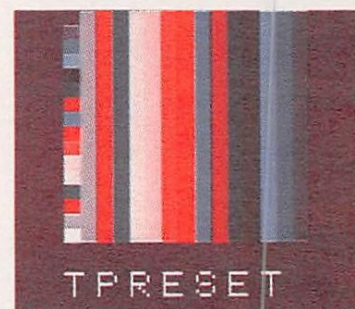
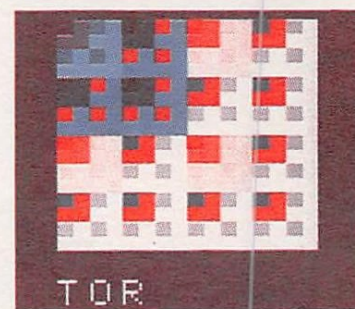
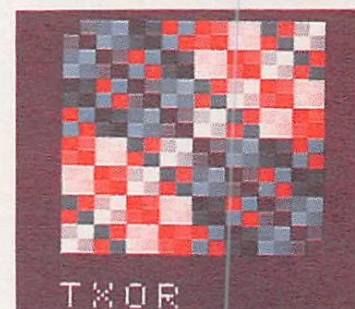
しかし、ことCOPY命令で論理演算子を使う場合には、いままでのグラフィック命令のそれとは違い、パッと世界が広がってくるのだ。

グラフィックの重ね合わせや次ページで紹介する必要な部分の切りぬきなど、実用性がグンと増してくる。

ここでは、論理演算によって、どの色が何色に変化するのかを紹介しよう。もっとも、いままでにも何回か紹介してきたことなので、きちんと理解している人は読み飛ばして構わない。

下のリスト1はCOPY命令で指定できる10種類の論理演算の結果を、すべて表示させるプログラムだ。実行すると、下の写真のような画面になるので、スペースキーを押してみよう。画面の右側に、右の写真のようなグラフィックと文字が表示されていくだろう。

付録ディスクでは、LIST1〜5．SBYというファイルの一部になっている。実行してみてほしい。

画面左上の横じまが元の状態。左下の縦じまが転送されるグラフィック。色コードを暗い灰色に変えている

PSET：元の状態に関係なく、そのままの色で表示される
PRESET：元の状態に関係なく、色コードが反転して表示される
OR：元の状態によって、色コードをORして表示される
AND：元の状態によって、色コードをANDして表示される
XOR：元の状態によって、色コードをXORして表示される

TPSET：透明色の部分は元の状態の色。ほかはPSETとおなじ
TPRESET：透明色の部分は元の状態の色。ほかはPRESETとおなじ
TOR：TORの場合、ORとまったくおなじ結果になる
TAND：透明色の部分は元の状態の色。ほかはANDとおなじ
TXOR：TXORはXORとまったくおなじ結果になる

■リスト1　論理演算のようすをみる

```
10 COLOR15,0,0:SCREEN5:OPEN"GRP:"AS#1
20 FORI=0TO15
30 LINE(0,I*4)-STEP(63,3),I,BF
40 LINE(I*4,100)-STEP(3,63),I,BF
50 NEXT:COLOR=(1,3,3,3)
60 FORI=0TO9:READ A$(I):NEXT
70 IFINKEY$<>" "THEN70ELSEA=(A+1)MOD10
80 LINE(100,100)-(200,200),0,BF
90 COPY(0,0)-(63,63)TO(100,100)
100 ON A GOTO120,130,140,150,160,170,180
,190,200
110 COPY(0,100)-(63,163)TO(100,100),,PSE
T:GOTO210
120 COPY(0,100)-(63,163)TO(100,100),,PRE
SET:GOTO210
130 COPY(0,100)-(63,163)TO(100,100),,OR:
GOTO210
140 COPY(0,100)-(63,163)TO(100,100),,AND
:GOTO210
150 COPY(0,100)-(63,163)TO(100,100),,XOR
:GOTO210
160 COPY(0,100)-(63,163)TO(100,100),,TPS
ET:GOTO210
170 COPY(0,100)-(63,163)TO(100,100),,TPR
ESET:GOTO210
180 COPY(0,100)-(63,163)TO(100,100),,TOR
:GOTO210
190 COPY(0,100)-(63,163)TO(100,100),,TAN
D:GOTO210
200 COPY(0,100)-(63,163)TO(100,100),,TXO
R
210 PSET(100,180),0:PRINT#1,A$(A):GOTO70
220 DATA PSET,PRESET,OR,AND,XOR,TPSET,TP
RESET,TOR,TAND,TXOR
```

●プログラム解説
10　画面の色、画面モード設定／グラフィック画面への文字表示準備
20　元絵作成ループ開始
30　元の状態用グラフィック作画
40　転送用グラフィック作画
50　ループ終了／パレット切り換え
60　論理演算子の名前データ読みこみ
70　スペースキー入力待ち／分岐用変数切り換え
80〜90　前回の表示消去／元の状態用グラフィック複写
100　各論理演算COPYへ分岐
110〜200　論理演算COPY
210　論理演算子の名前表示／行70へ
220　論理演算子の名前データ

 SB進路相談

⇨グラフィック画面のときに、参照されるROMのデータの代わりを、自分で用意してやればいいのだ。とはいっても、そうやすやすとはいかないだろう。そこで、前に『新・マシン語の気持ち』の担当者がグラフィック画面でのキャラクタパターン定義プログラムを作って持っていたハズなので、来月の付録ディスクにサンプルとして入れてもらうことにしよう。それでいいよね、清水くん。

COPY命令と論理演算

COPY命令の使い方はもうわかっただろうか？　ここではCOPY命令を使ったいろいろなテクニックを紹介しよう。

■グラフィックの拡大

右のリスト2を見てほしい。このプログラムは、画面の左上4分の1のグラフィックを、縦横2倍にして、全画面の大きさに拡大するものだ。

このCOPY命令を使っての拡大の方法は、このほかにもいくつかのやり方があるが、ここではスピードは遅いが、派手な方法を選んで掲載した。

実際に拡大している部分は、行100からのサブルーチンなので、いろいろ試してみよう。

■合わせ鏡の原理

リスト3は2つのCOPY命令を使って、たった4ドットのグラフィックを元に、タイルパターンで画面をうめるものだ。

転送元のグラフィックに転送先の座標を重ねることで、グラフィックを作成する方法だ。

リスト4もそういったやり方で、8×8ドットのパターンで画面をうめるものだ。

これらのプログラムで使われているテクニックは、合わせ鏡の原理を元にしているのだ。

■グラフィックの切りぬき

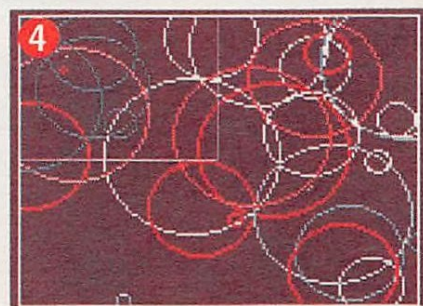
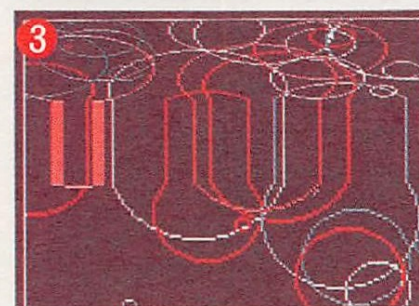
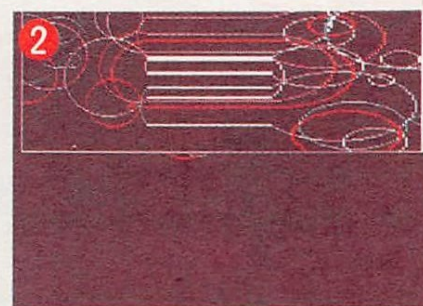
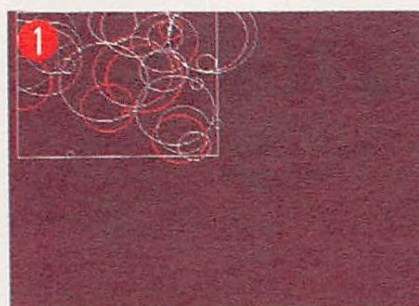
最後に紹介するのがリスト5のグラフィックの一部を切りぬく方法だ。

写真では丸い形にしているが、例えば6角形でもいいし、自由線で囲んだ形で切りぬくことだってかんたんにできる。

これは、論理演算子のANDを使った基本的なやり方だ。

次回のSB講座では、ファイルや配列変数とグラフィックとのCOPY命令を紹介しよう。

■リスト2　拡大

```
10 COLOR15,0,0:SCREEN5:DEFINTA-Z
20 FORI=0TO30
30 X=RND(1)*128:Y=RND(1)*106
40 R=RND(1)*30:C=RND(1)*14+2
50 CIRCLE(X,Y),R,C
60 NEXT
70 LINE(0,0)-(127,105),15,B:BEEP
80 IF STRIG(0)=0 THEN80 ELSE GOSUB100
90 IF STRIG(0) THEN90 ELSE70
100 FORI=127TO0STEP-1:A=I*2
110 COPY(I,0)-(A,105)TO(I+1,0)
120 NEXT
130 FORI=105TO0STEP-1:A=I*2
140 COPY(0,I)-(255,A)TO(0,I+1)
150 NEXT:RETURN
```

↑写真①が元のグラフィック。ビープ音が鳴ったらスペースキーを押してみよう。②は横に2倍しているところ。③で縦も2倍にしている。④で拡大が終了し、また拡大を繰り返す

■リスト3　合わせ鏡の原理

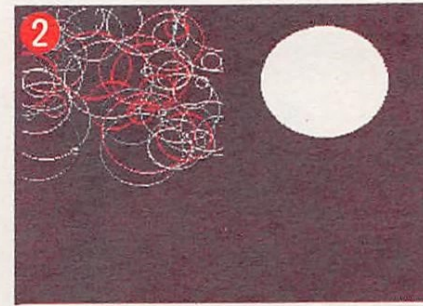
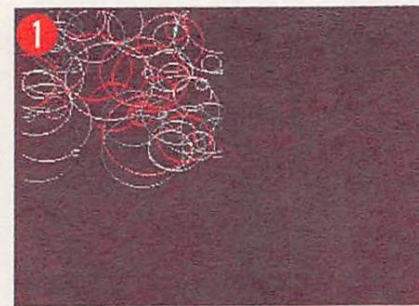
```
10 COLOR15,0,0:SCREEN5:DEFINTA-Z
20 FC=8:BC=15
30 PSET(0,0),FC:PSET(1,0),BC
40 PSET(0,1),BC:PSET(1,1),FC
50 COPY(0,0)-(253,1)TO(2,0)
60 COPY(0,0)-(255,209)TO(0,2)
70 BEEP
80 IF STRIG(0)=0 THEN80
```

■リスト4　パターンでうめる

```
10 COLOR15,0,0:SCREEN5:DEFINTA-Z
20 FORI=0TO7:READ A$
30 FORJ=0TO7
40 C=VAL("&H"+MID$(A$,J+1,1))
50 PSET(J,I),C
60 NEXT:NEXT
70 BEEP:GOSUB120
80 COPY(0,0)-(247,7)TO(8,0)
90 BEEP:GOSUB120
100 COPY(0,0)-(255,203)TO(0,8)
110 BEEP:GOSUB120:END
120 IF STRIG(0)=0 THEN120
130 IF STRIG(0) THEN130 ELSE RETURN
140 '
150 DATA A32C23AF
160 DATA 32C23AFA
170 DATA 2C23AFA3
180 DATA C23AFA32
190 DATA 23AFA32C
200 DATA 3AFA32C2
210 DATA AFA32C23
220 DATA FA32C23A
```

①これが8×8ドットの元絵。リストの行150以降のデータが、各ドットのカラーコード(16進文字)になっているので、パターンを作り変えてもおもしろいかも。ビープ音が鳴るので、スペースキーを押そう。②横に複写していく。いったんビープ音が鳴って、スペースキーの入力待ちになる。③縦にも複写していき、画面がパターンでうめられる。スペースキーで終了する

■リスト5　切りぬき

```
10 COLOR15,0,0:SCREEN5:DEFINTA-Z
20 FORI=0TO60
30 X=RND(1)*128:Y=RND(1)*106
40 R=RND(1)*30:C=RND(1)*14+2
50 CIRCLE(X,Y),R,C
60 NEXT
70 LINE(128,0)-(255,105),0,BF
80 BEEP:GOSUB150
90 CIRCLE(192,53),40,15
100 PAINT(192,53),15
110 BEEP:GOSUB150
120 COPY(0,0)-(127,105)TO(128,0),,AND
130 BEEP:GOSUB150
140 END
150 IF STRIG(0)=0 THEN150
160 IF STRIG(0) THEN160 ELSE RETURN
```

①行20〜60で作成した、元絵のグラフィック。作成が終わるとビープ音が鳴るのでスペースキーを押そう。②行90〜100で切りぬく形を作成する。例えばここを3角形にしたりして試すのもおもしろい。ここもビープ音が鳴ったらスペースキーを押す。③元絵を切りぬく形に論理演算子のＡＮＤを使ってＣＯＰＹする。すると、白い切りぬく形の部分だけ、グラフィックが残るのだ

【ＳＢ講座よりお知らせ】 次回のＳＢ講座では、まだ作りかけだったＳＣＲＥＥＮ５のグラフィックエディタの完成版を、付録ディスクに収録する予定だ。はっきりいって、まだ、ぜんぜん手を付けていないので、どんなものに仕上がるかはわからないが、使いやすさだけを追求するつもりでいる。あまり使いたくはないが、もしかすると、少しマシン語を使うかもしれない。まあ、とりあえず、期待していてほしい。

スーパー ビギナーズ

超初心者 Super Beginners' 講座

第23回
グラフィック入門その11

COPY命令を使えば、かんたんにグラフィックを別の場所に移動することができる。今回は、グラフィックをディスクや配列変数に複写する方法を紹介する。

ディスクや配列とのCOPY

前回は、グラフィック画面で使うCOPY命令の、基本ともいうべきVRAM内でのCOPY命令を紹介した。

今月はグラフィックとディスクまたは配列変数とのCOPY命令を紹介しようと思う。

■VRAMからディスクへ

グラフィックをディスクに複写するというと、あまりピンとこないかもしれないが、つまりこれは、COPY命令を使ったグラフィックの保存と思っていれば間違いない。

グラフィックを保存する命令には、以前紹介したBSAVE命令というのがあるが、これは画面いっぱいを使ったCGなどを保存するのに有効だ。そしてCOPY命令を使った保存は、グラフィックの一部を長方形の形で保存するものだ。もちろん全部を保存することもできる。

このCOPY命令の書式は下表のとおりで、グラフィックの転送元領域の指定は、VRAM内でのCOPYとおなじだ。

COPY命令では、VRAM、つまりグラフィックを転送元とする場合、その指定のしかたはおなじなので覚えておこう。

COPY命令でディスクに保存したグラフィックは、やはりCOPY命令を使わないと読みこむことができない。だから、ファイル名は、COPY命令で保存したグラフィックだとすぐわかるように、例えば「.GRP」というように、拡張子を決めておくといいだろう。

■ディスクからVRAMへ

ディスクからVRAMにグラフィックをCOPYする場合の書式も下表のとおりだ。

ここでも転送先がVRAMの場合には、その指定方法はおなじなので覚えておこう。

どちらの場合も、ファイル名は文字変数を使って指定することができる。

また、ファイル名に「A：」のようにドライブ番号を付けてやれば、どのドライブかを指定することもできる。

ターボRを使う場合、ドライブ名を「H：」としてやればRAMディスクとでCOPYできるので、速くて便利だ。

■配列変数とのCOPY

見出しをみて、不思議に思う人もいるんじゃないかな？

COPY命令では、配列変数にグラフィックを複写することもできるのだ。

どんなことに使えるのかって聞かれると、少し困ってしまうけれど、VRAM内でCOPYするよりも速く複写できるので、例えば、RPGなどのマップのように、小さなグラフィックをたくさん組み合わせて、大きなグラフィックを作るときなどに有効だろう。

ただし、グラフィックが複写できる配列は、数値変数でないといけないので注意。文字変数の配列だと、ファイル名と解釈されてしまうのだ。

■VRAMとディスクとのCOPY命令書式

COPY【転送元領域】TO【ファイル名】

VRAMの【転送元領域】で示された領域を、ディスクに【ファイル名】で保存する。

●【転送元領域】の指定方法
・（転送元開始点）－（転送元終了点），図形のあるページ
・（転送元開始点）－STEP（終了点までの増分），図形のあるページ

●【ファイル名】
・「ファイル名」＋「拡張子」

COPY【ファイル名】TO【転送先】【，論理演算子】

ディスクから【ファイル名】のグラフィックを、VRAMの【転送先】で示された位置に読みこむ。

●【転送先】の指定方法
・（転送先開始点），転送先のページ，論理演算子

SB進路相談

【SB講座よりお知らせ】 今月の付録ディスクに収録する予定だった、SCREEN5エディタの完成版は収録できませんでした。ゴメン。AVフォーラムとCGコンテストの新しい付録ディスク用のプログラムを作っていたためと、忙しい時期に性悪な風邪をひいて寝こんでいたために完成することができなかったのだ。来月こそは付録ディスクに収録するので、もう少し待っていてほしい。

ここでは、VRAMと配列変数のCOPY命令について紹介しようと思う。

■配列変数を宣言する

配列変数とグラフィックとのCOPY命令では、まず必要な大きさの配列変数を、DIM文で宣言しておく必要がある。

用意する配列変数の大きさは、右表のように計算して出す。

あまり大きなグラフィックを複写しようとして、大きな配列変数を宣言すると、メモリが足りなくなってエラーが出ることがあるので注意しよう。また、右表の計算で得られた大きさはおよその値なので、メモリを節約したい人は自分で調整しよう。

整数型の配列を使うほうが、いろいろな意味で有利だぞ。

■配列変数とのCOPY

VRAMから配列変数に複写するCOPY命令の書式は、下表のとおりだ。

転送元領域の指定は前ページで説明したように、VRAMの場合はいつでもおなじだ。

転送先は配列変数名となっているが、例えば右表のように宣言した配列に複写するのなら、ここには「A%」とだけ指定すればいい。注意してほしいのは配列変数のグループ1つにつき、1つのグラフィックしか複写できないということだ。だから、2次以上の配列を使って、添字でグラフィックを使い分けようということはできないぞ。

配列変数からVRAMに複写するCOPY命令の書式も下表にあるとおりだ。

ここでも転送先がVRAMの場合の指定方法は、いままでとおなじなのだが、転送元の指定は配列変数名と方向を指定することになっている。

この方向の指定は、転送先の座標から、どの方向へ向かってグラフィックを複写するかを表していて、0〜3の数値で指定する。各数値とその複写方向については下表を参照しよう。

この方向をうまく使えば、グラフィックを左右や上下にかんたんに反転できるので、試してみてほしい。

■配列変数の大きさの計算方法

①まず、配列変数にCOPYするグラフィックの大きさを計算しておく。

【計算式】　(x1, y1)−(x2, y2)

⇨　(x2−x1+1)

×(y2−y1+1)

例：(0, 50)−(150, 100)の場合

=((150 −0+1)×(100 −50+1)

=　151 ×51　=　7701

②計算した値を、画面モードと変数の型により下表から得た数値で割る（端数は切上げる）。

	整数型の配列	単精度実数型の配列	倍精度実数型の配列
SCREEN8以降	2	4	8
SCREEN5または7	4	8	16
SCREEN6	8	16	32

例：SCREEN5で前述の範囲を整数型の配列に複写する場合　7701÷4 =1926　(端数は切上げ)

③2を加えてDIM文で配列を定義する。

例：DIM　A%(1928)

■VRAMと配列変数とのCOPY命令の書式

COPY【転送元領域】TO【配列変数名】

VRAMの【転送元領域】で示された領域を、【配列変数名】で指定された数値配列に複写する。

●【転送元領域】の指定方法

・(転送元開始点)−(転送元終了点), 図形のあるページ

・(転送元開始点)−STEP(終了点までの増分), 図形のあるページ

●【配列変数名】の指定方法

・配列変数の変数名のみを指定する

COPY【配列変数名】【, 方向】TO【転送先】【, 論理演算子】

【配列変数名】で指定された配列から、VRAMの【転送先】で示された位置にグラフィックを複写する。また、【方向】を指定することにより、元の絵を左右、上下で反転させることができる。

●【転送先】および【論理演算子】の指定方法

・(転送先開始点), 転送先のページ, 論理演算子

●【方向】の指定方法

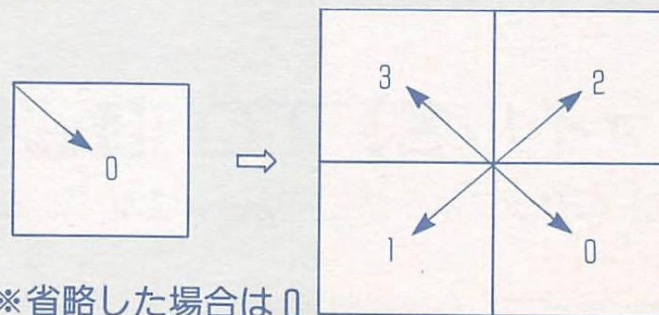
0=転送先の位置から右下へ複写される

1=転送先の位置から左下へ複写される

2=転送先の位置から右上へ複写される

3=転送先の位置から左上へ複写される

※省略した場合は0

SB進路相談　【先月の質問：グラフィック画面でのキャラクタパターン定義のやり方について】先月のSB進路相談で「来月の付録ディスクにサンプルを〜」と書いていたが、じつは収録していない。それは、マシン語の気持ち担当者の馬場（彼は今、大学院でとてもハマっていて忙しい）への連絡が遅れ、サンプルが手に入らなかったので、自分で作ろうと思っていたところに、BASICピクニック担当者のコルサコフが⇨

COPY命令の用途

ここでは2回にわたって紹介した、グラフィック画面に関係したCOPY命令の、代表的な使われ方について紹介しよう。

■VRAMやディスクを使ってアニメーション

COPY命令の使い方については前回でもいくつか紹介していたが、まずはじめに紹介するのは、より使われる頻度の高いアニメーションでのCOPY命令の使われ方だ。

右の2つの連続写真は、以前BASICピクニックで誌面に掲載された、アニメーションのサンプルのものだ。今月の付録ディスクにサンプルとして、

ANIME．SBZ
FILE．SBZ

というファイル名で収録してあるので実行してみよう。

まず「VRAM～」のほうは、連続写真の下の大きめの写真にあるような元絵が、ページ1～3に描かれる。これをCOPY命令を使ってひとつずつ、連続してページ0の画面の中央に複写していくのだ。

実際の画面には、ページ0が表示されているので、アニメーションしているように見える。

「FILE～」のほうは、元絵を作画してディスクに1枚ずつ保存し、アニメーションさせるときはこれを1枚ずつディスクから読みこんで表示するのだ。

どちらの場合にも、まずアニメーションの元絵を作成して保存しておき、そこからCOPY命令で1枚ずつ連続表示させてアニメーションさせるのだ。

■BASICでかんたんにマップが作れる

下の2枚の写真はOrcに頼んで作ってもらったHEXマップ表示サンプルだ。

付録ディスクには、

HEXMAP．SBZ

というファイル名で収録してあるので、実行してみよう。

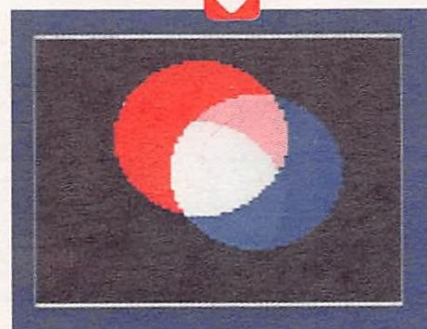
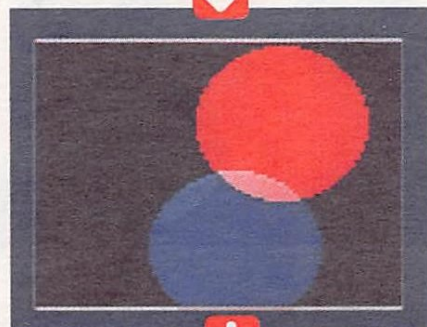
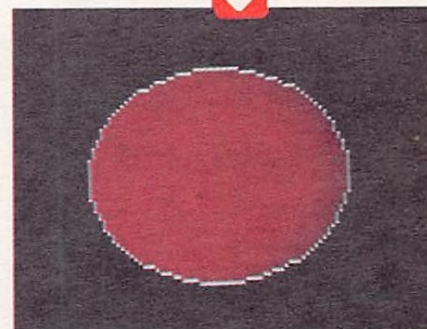
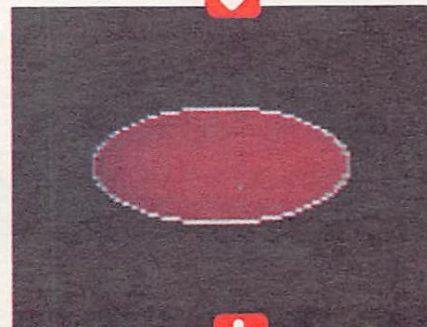
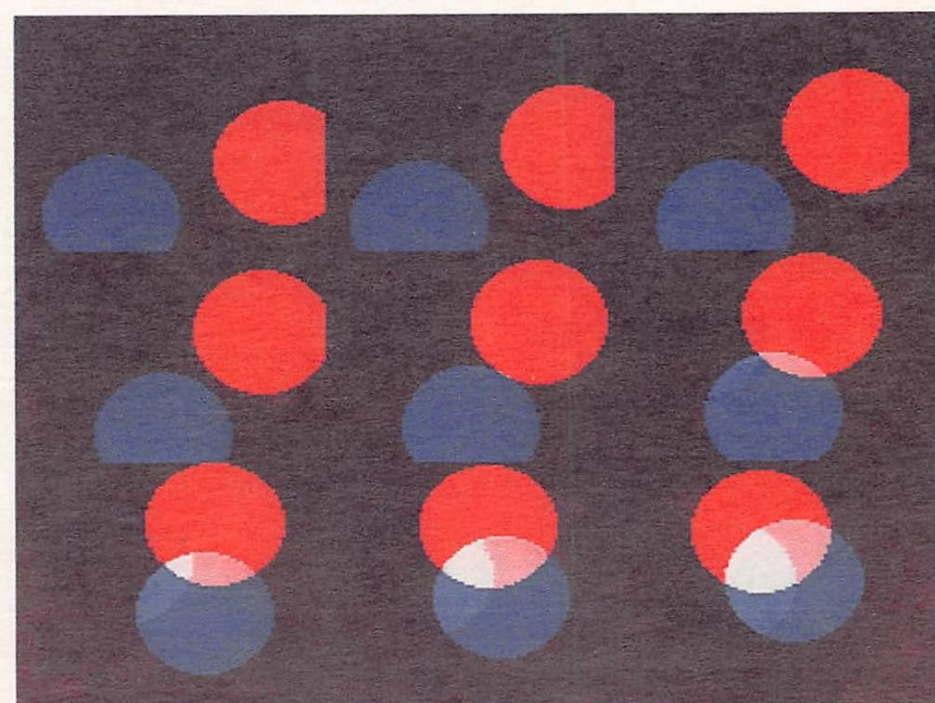
このサンプルでは、まず右下の写真にあるような、マップを構成する元絵を作画し、データ(リストの後半)をもとにそのグラフィックを順に、論理演算子のTPSETを使ってCOPYしてマップを作成している。

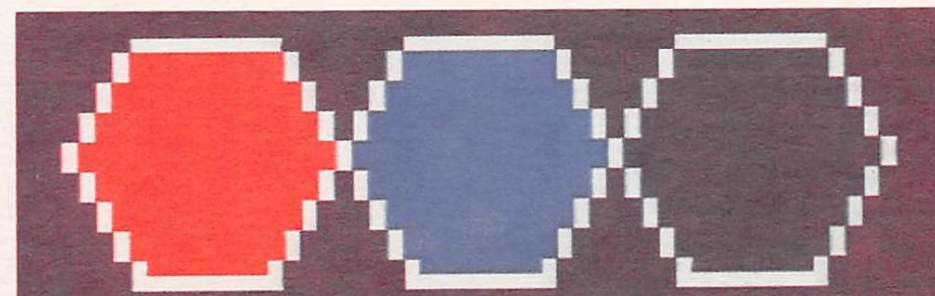
地形は3種類しかないが、BASICでシミュレーションのHEXマップを表示することが、短いプログラムでも可能だということがわかるだろう。

もちろん、RPGのような四角いマップならもっとかんたんに作れるだろう。

地形の種類を増やしたり、マップの形や大きさを変えて試してみてほしい。

⬆自分でシミュレーションを作ってみたい、なんて思ったことがある人はけっこういるハズだ。このマップはBASICのかんたんなプログラムで表示されているので参考にしてみよう

VRAMから　ディスクから

⬆左が「ANIME.SBZ」、右は「FILE.SBZ」の実行画面。「FILE.SBZ」はターボRなら行20のREM文を取るとRAMDISKで動き、実行後はGOTO90でリプレイする

⬆「ANIME.SBZ」のページ1の画面。このように、あらかじめアニメーションの元絵を作画しておき、それをCOPY命令で複写してきてアニメーションさせるのだ

たとえば、こんな絵で……

⬆「HEXMAP.SBZ」では、この写真のようなかんたんな3つの地形しか表示しないが、自分で地形を増やしたり、もっと凝ったグラフィックにしても面白いぞ

「テーマが大きいので、1月号のBASICピクニックで取り上げる」といってくれたからだ。以前からこの質問は多いのだが、BASICのワークエリアが関係し、マシン語を使わないと実用的でないので、ファンダムでは今まで答えたことがなかった。そこで、欄外やカコミなどの小さなスペースではなく、きちっと誌面で答えようということになったのだ。そういうわけで、福岡の清水くん、もう少し待ってほしい。

SUPER BEGINNERS' 講座

スーパー ビギナーズ

超初心者

新装開店

第1回
新しい出発のためのミニ用語集

新装開店したSB講座では、「マニュアルよりもわかりやすく」をモットーに、あらためてプログラミングの基礎を紹介していく。今回は、まず用語について。

「汎用」の意味がわからず苦しんだ日々

初心者のうちは、プログラムの「プ」の字もできなくてあたりまえだ。なにも恥じることではない。胸を張っていばれることでもないが。

その初心者が自分でプログラムを組めるようになるには、ただひたすら基本的な知識を身につけ、とにかくたくさん考えることが必要だ。

そして、ちょっとくらいつまずいたからといって、かんたんにあきらめないことが大切だ。いろいろ調べたり試したりして、自分なりに納得がいくまでやってみてから結論を出そう。そうしないと、なかなか上達できないで、かえってつらいぞ。

ところで、プログラムを組むのに必要な基本的なこととはなんだろうか。それは、命令のつづりや小手先のテクニックなどではなく、手に入れた情報が理解できるように、用語などをきちんと覚えておくことだ。

ファンダムで毎月のように紹介されている内容が、言葉の意味がわからないだけでムダになるのはもったいない。

用語のすべてをここで紹介できるわけではないし、説明も十分ではないが、今回の記事が解説を理解する最初のきっかけになってくれればと思う。

*　*　*

変数と定数

変数は内容が自由に変更できる箱のようなもので、アルファベットで始まる英数字で表される(ただし最初の2文字しか区別されない)。それに対して、定数とは

A=123

の123のように具体的な数値や、

A$="ABC"

の"ABC"のように具体的な文字列のことをいう。ただし、形式上変数であっても特定の値以外にはならないものを「プログラム定数」ということがある。

汎用変数

汎用(はんよう)とは広く用いられるという意味で、プログラムの中で決まった役割を持たずに使われる雑用係的な変数のことをいう。雑用といっても、その時々では意味があるので、汎用変数の多いプログラムはわかりにくく、混乱のもとにもなる。ちなみに「汎用」という言葉自体もむずかしい。私も編集部に入るまえの読者時代、この言葉の意味がわからず苦しんだものだ。

カウンタ

値が一定の割合で増加もしくは減少する変数のことで、例えば10回当たったらクリアなどというときの、当たった回数を数える場合などに使用する変数のこと。通常「C=C+1」などの形で使われている。

ループ

ひとつのFOR文から、それに対応するNEXT文までのことをループといい、FOR文で指定されたカウンタ用の変数を、ループ用変数と呼んでいる。ループ用変数は、そのループの回数を管理している変数なのだ。

フラグ

プログラムの中で状態を判別するときに使用する変数。ふつうは判定の結果をほかの処理に影響させるために、その判定結果を保存しておくのに使用する。

入力

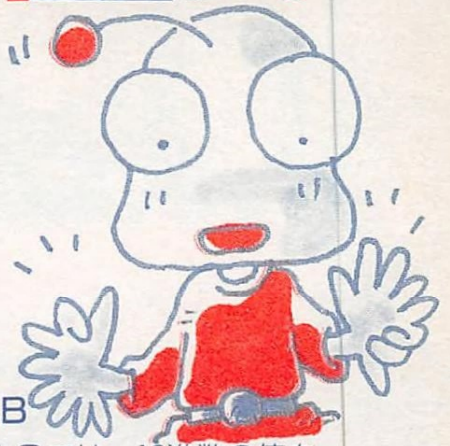
入力とは外部からコンピュータに信号を送ることをいう。また、これらの信号を受け取る関数として、カーソルキーならSTICK関数、スペースキーならSTIRG関数などが用意されている。

入力用変数

入力用変数は、上記のSTICK関数などによって入力された値を保存しておくのに使われる。例えば、

S=STICK(Ø)
T=STRIG(Ø)

などの、S、Tがそれだ。入力

今月から新装開店したSB講座。SB進路相談はとりあえず打ち切りにしたが、まだこの広いスペースの使い道は、はっきりとは決まっていない。いまのところ、SB進路相談のように質問に答えていくかどうかはわからないが、誌面では紹介できないような、ちょっとおもしろいテクニックやお遊びを紹介していこうと思っている。問題は、毎月これだけのスペースが埋められるほど、ネタがあるかどうかなんだけど。そうそう、質問は引き続き受け付けるので、SB進路相談がなくなっても遠慮せずに送ってきてほしい。ただ、欄外では答えられない質問が多いので、たぶん、ここに

これでプログラム解説がわかるようになる

用関数はつねに入力を受け付けているので、そのときどきで値が変化してしまう。そのためにいったん入力用変数に保存して、座標計算などの処理で不都合が起きないようにしているのだ。

添字(そえじ)
添字とは、配列変数のカッコの中に入れる数値のことだ。変数の意味の解説では、nなどのアルファベットの小文字を使って表現している。

数字と数値
紙に書かれた「123」という数は、文字であり数値でもあるが、MSXでは数字(文字列)と数値がはっきり区別されている。BASICでは、おもに
"123"
と書くと数字で、そのまま、
123
と書くと数値になる。もちろん、MSXの内部では数字と数値の記憶のされ方はまったく違う。

キャラクタ
ふつうは画面に表示される文字のこと。例えば、「キャラクタパターン定義」なら、文字の形を登録することを意味し、「キャラクタコード」なら、文字のひとつひとつに付けられた番号のことをいう。ただし、場合によっては、アニメの「キャラクタ」などと同様にゲーム中の登場人物などを指すこともある。

太文字処理
キャラクタパターン定義のひとつで、もともと文字の形を1ドットずらして二重にし、太い文字の形を作ること。そこから、一般的に一定の規則に基づいて文字の形を変更する処理全般も指す傾向がある。具体的に斜体処理などと表記することもある。

初期設定
初期設定とは、プログラムが実行されたとき、使用する画面を準備したり(画面初期化)、変数の型や配列変数の宣言、おもな変数の始めの値を設定したり(変数初期化)する、始めのときにだけ実行される部分のことをいう。プログラムは通常、初期設定部、メインルーチン、サブルーチン、データの4ブロックにわけられて整理されている。

メインルーチン
その名のとおり、プログラム全体のメイン(主要部分)となるところ。プログラム実行中は、ふつう、このメイン部分を繰り返し実行している。

サブルーチン
メインに対して補助的な役割を果たす部分をサブルーチンとか、サブという。一般的にメインルーチンからGOSUB命令で呼び出されて実行され、処理が終わるとRETURN命令でもとの場所にもどる。サブルーチンをうまく使えば、プログラムが整理されて見やすくなる。

データ
初期設定で変数に設定する値や、スプライトやキャラクタに設定するパターンなどの定数や、表示する文字列などをまとめてある部分をいう。具体的には、DATA文のあとにそれらを並べておき、READ文で読み出す。

マシン語
マシン語とは、MSXなどのコンピュータにとっての母国語で、人間にとってはわかりにくい言葉だ。BASICが「英語」のような言葉なのに対して、マシン語は、数字ばかりの「暗号」のようなものだと思えばよい。

USR関数
マシン語はBASICとは違う言葉なので、おなじに扱うことができない。そのために、BASIC中でマシン語を呼び出すための命令がUSR関数なのだ。

RAMとVRAM
RAMとは、プログラムや変数の値などが保存される領域で、VRAMは画面表示に関する情報を保存する領域だ。スプライトの形や色、文字の形などもVRAMに保存されている。似たような名称なので、よくおなじものと誤解されるが、RAMとVRAMはまったく違うものだ。

マシン語領域の確保
マシン語はRAMに直接書きこまれて実行されるので、その場所をあらかじめ確保すること。また、おなじようなものに文字領域の確保というのがあるが、これは文字列を連結したり、文字変数を扱うときに使われる領域を確保することだ。どちらもプログラムの始めにCLEAR命令によって設定される。

POKE
POKEはRAMに値を書きこむ命令で、おもにマシン語の書きこみで使われる。似たものに、RAMから値を読みこむ関数として、PEEKがある。

VPOKE
VPOKEはVRAMに値を書きこむ命令で、キャラクタやスプライトのパターン定義や色の設定で使われることがある。これにもVRAMから値を読みこむ関数としてVPEEKがあり、太文字処理などでよく使われる。

16進数
ふだん使っている数は、10進数と呼ばれるもので、10までいくとひとつ桁が上がるのでこう呼ばれている。同様に、16進数は16までいくとひとつ桁が上がる数のことだ。10までは10進数とおなじで、11～15にはアルファベットのA～Fの6文字が割り当てられている。また、2まで行くとひとつ桁があがる2進数という数もある。どちらもコンピュータと相性のよい数なのだ。

&H、&B
BASICでは、16進数の値を表現するときには、先頭に「&H」を付けて表記する決まりがある。これは、例えばプログラムに1234という数値があったとき、これが10進数なのか16進数なのかMSXにわかるようにするためだ。おもにRAMやVRAMのアドレスを表現するときに使う。ちなみに、2進数の場合もおなじように、先頭に「&B」を付ける決まりがある。なお、Simple ASMなどのアセンブラでは16進数を「1234H」のように、最後に「H」を付けて表現し、2進数を「1010B」などのように最後に「B」を付けて表現する。

アドレス
番地ともいう。これは、RAMやVRAMなどの、どの部分かを示す番号のことだ。アドレスは16進数で示すことが多い。

＊　＊　＊

ファンダムに掲載された作品は、基本的にプログラムの解説が施されている。これらの解説では、今月紹介した用語が使われているので、今まで読んでもわからなかったという人は、これらの用語をふまえてもう一度読んでみよう。今までよりは内容がわかるようになったはずだ。

■次回以降の予定
次回から、いよいよプログラムの組み方を紹介していこうと思う。具体的には、文字入力処理や座標計算処理といった、処理単位でのプログラム講座にする予定でいる。
処理単位にしておけば、自分のプログラムでサブルーチンとして使うことも可能だろう。乞う、ご期待。　(MORO)

は載らないだろう(と思う)。▷▷▷今月紹介した用語以外にも、超初心者の理解の妨げになっている言葉はたくさんあると思う。本文中での用語解説は、今月だけだが、この欄外のスペースで「これも教えてほしい」とか「この意味がまだよくわからない」といった意見も受け付けるので、どちらも「Mファン　SB講座」あてにどんどん送ってきてほしい。こういうのは利用しないと損だぞ。必ずハガキまたは封書で編集部に送ってほしい。ディスクはダメだぞ。

SUPER BEGINNERS' 講座

スーパービギナーズ

超初心者

第2回
今月のテーマ：
スプライトを動かす

ファンダム掲載作品でよく見かける、ＳＴＩＣＫ関数を使った入力と、スプライトパターンを切り換えてのアニメーションを、組み合わせて紹介するぞ。

スティック入力値とその方向

自分のプログラムで、ジョイスティックを使いたいのだけど、その方法がわからない、なんていう人、けっこういるんじゃないかな？　最近マウスゲームが多くなっているのは、じつはそのせいだったりして。

■ＳＴＩＣＫ関数とは

ジョイスティックはおろか、カーソルキーにも対応している入力用の関数がＳＴＩＣＫだ。

これは、押されている方向を調べるための関数(図１参照)で、Ｓ＝ＳＴＩＣＫ(入力機器番号)のように使う。このときの入力機器番号とは、０ならカーソルキー、１ならポート①のジョイスティック、２ならポート②だ。

■入力から座標へ

ＳＴＩＣＫ関数の使いにくい部分は、返ってくる値、つまり入力値から座標に変換するのが面倒なところにある。

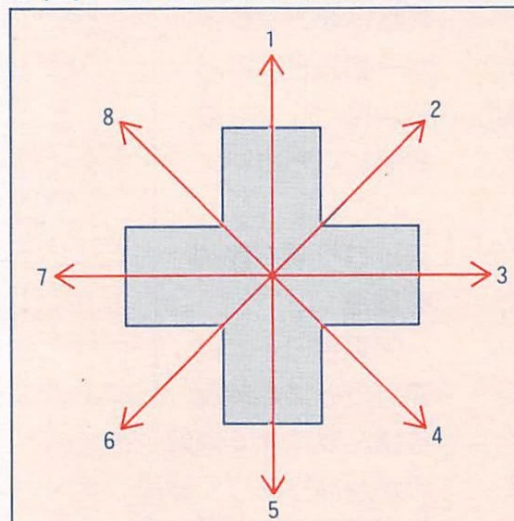
例えばもっとも単純に、ＩＦ文を羅列(られつ)して０〜８のすべての場合について判定するとしよう。しかしこれでは処理が長くなり、画面の大半が埋まってしまい、とてもうっとうしい。

そこでファンダムの標準的な使われ方を３つ紹介しよう。

右のリスト１とリスト２は、関係演算子を使って、直接座標計算に取りこんだ例だ。入力用の変数Ｓが１だったら、という具合に、仕組みは自分で考えてみよう(欄外参照)。

また、よく座標調整を一緒にしていることがあるが、これは図２のようにして別に行おう。

リスト３はあらかじめ配列変数に０〜８の入力値に対応する増分を設定しておき、座標計算で使用する例だ。これは増分を自由に設定できるし、計算も速くすっきりしていていい。

●図1 ＳＴＩＣＫ関数の値と方向との関係

ＳＴＩＣＫ関数は、実行された瞬間のカーソルキーまたはジョイスティックの状態を調べて、押されている方向を０〜８の数値で返す関数だ。この値と押されている方向との対応は、左図のように上から時計回りで１〜８の値になり、押されていないときは０になる。

ＳＴＩＣＫ関数の値は、リアルタイムで変化している(欄外参照)。だから、座標計算などで使用するときには、Ｓ＝ＳＴＩＣＫ(０)のようにして、いちど変数に入力値を保存しておき、計算では保存した値を参照するようにしよう。

●リスト1 4方向の増分計算例

```
X=(S=7)-(S=3):Y=(S=1)-(S=5)
```

●リスト2 8方向の増分計算例

```
X=(S>1)*(S<5)+(S>5)
Y=-(S>0)*(S<3)+(S>3)*(S<7)+(S>7)
```

●リスト3 増分用の配列変数を使う

```
10 DIM X(8),Y(8)
20 FOR I=0 TO 8:READ X(I),Y(I):NEXT
30 S=STICK(0):X=X+X(S):Y=Y+Y(S)

100 DATA 0,0, 0,-1, 1,-1, 1,0, 1,1, 0,1,
 -1,1, -1,0, -1,-1
```

●図2 座標の調整方法

	制限範囲の限界の位置で止める	反対側から出現させる(ループさせる)
IF〜THEN文を使う場合	IF X<0 THEN X=0 ELSE IF X>255 THEN X=255 Xが０より小さいとき、つまり画面の左端から出たときはTHEN以降でXを０に設定する。出ていないときは、Xが255より大きいか判定し、大きいときはXを255に設定している。つまり、変数Xの値が０〜255という範囲から出たとき、限界の値に設定して超えないようにしているのだ。	IF X<0 THEN X=X+256 ELSE IF X>255 THEN X=X-256 Xが０より小さければ、Xに256を足す。例えば−１なら255になる。反対にXが255より大きければ、Xから256を引く。256なら０になるのだ。すると、まるで左端と右端がつながっているかのように、左端から出ると右端から、右端から出ると左端から出現するようになるのだ。
計算式だけで行う場合	X=X+(X<0)*X+(X>255)*(X-255) 上のＩＦ文で行っている処理を、関係演算子を使って行うと、このようになる。Xが０より小さければ、Ｘ＝Ｘ−Ｘで０になり、Xが255より大きければ、Ｘ＝Ｘ−Ｘ＋255でXは255になる。ＩＦ文のほうが速いが、分岐してしまうので、行末に置かなくてもいい部分で使いやすい。	X=(X+256)MOD256 範囲内でループさせる場合、こんなかんたんな計算式で済んでしまう。ＩＦ文を使う場合と比べてとてもコンパクトだ。MODというのは、割り算した余りを求める演算子で、９MOD４なら１が答えになる。Xが０より小さいときと255より大きいときでどうなるか、ちょっと考えてみよう。

【STICK関数】STICK関数の入力値はリアルタイムで変化する。「ＦＯＲＩ＝０ＴＯＩ：Ｉ＝０：ＬＯＣＡＴＥ０，０：ＰＲＩＮＴ　ＳＴＩＣＫ(０)：ＮＥＸＴ」と打ちこんでリターンキーを押そう。画面の左上にSTICK関数の値が表示されるので、カーソルキーを適当に押してみよう。ほら、すぐ反応するだろう？　【関係演算子】関係演算子とは、大きい、小さい、等しいとかの大小関係の結果を、０か−Ｉの数値で返す演算だ。例えば、「Ｘ＜０」という演算では、この関係が成り立つ、つまりＸが０より小さいなら−Ｉ、そうでなければ０になるのだ。

スプライト表示とその工夫

今月もファンダムのいくつかの作品では、ゲームに登場するキャラクタが、なにやら動いているものがある。

この場合、動くといっても、右や左に動いていることをいっているのではなく、表示されたキャラクタそのものが、ちょこまかとアニメーションしていることをいっているのだ。

いったいこれは、どうやっているのだろうか。

■スプライトがアニメするわけ

このアニメーションしているキャラクタは、スプライトで表示されていることが多い。

スプライトを表示する命令にPUTSPRITEというのがあるが(欄外参照)、これはたんにスプライトを表示するための命令なので、アニメーション表示の機能などはない。

じつはこういったスプライトのアニメーションは、あらかじめ何枚かのスプライトパターンを作っておき(下のカコミ参照)、スプライトを表示するときに、それらのパターンを切り換えて表示しているのだ。そのとき、どのパターンを表示するかを決めているのが、変数の地味な計算の部分なのだ。

■変数の切り換えのいろいろ

例えば、0と1の2つのスプライトパターンを、交互に表示したいというとき、値が0と1で切り換わる変数を用意すればOKだ。PUTSPRITEで画面に表示するときに、その変数を使ってパターン番号を指定するのだ。

図3のいちばん上のように、0と1で変数を切り換えるなら、

A＝1－A

とすればかんたんに切り換えることができるのだ。

変数を切り換えるとき、計算式を使うと複雑になってわかりにくい場合がある。そのようなときは、あらかじめ配列変数に必要な数値を設定して、カウンタ用の変数を使って、その配列から数値を得る方法もある。

図3にはこれらを含むいくつかの方法を紹介しておいたので、覚えておいてほしい。

●図3 変数の値を切り換える

●変数を2値で切り換える

2値で切り換えを行いたい変数に、どちらかの値を設定しておき、切り換えるときは、

変数＝2値の合計－変数

とすれば、毎回値が切り換わる。

例：変数Aを－2と3で切り換える(変数Aには片方の値を設定)

A＝(－2＋3)－A　⇨　A＝1－A

●範囲内で増加(減少)する

変数の値を増加(減少)させていき、一定の値で止める場合には、関係演算を使うとよい。

例：変数Aの値が10になるまで1つずつ加算する。

A＝A＋－1＊(A＜10)　⇨　A＝A－(A＜10)

●変数の値を循環させる

変数の値を、例えば1、3、5、7、1、3……のように、循環させたい場合には、MODを使うとよい。

例：変数Aを1、3、5、7で循環させる(ただし、変数Aにはいずれかの値を設定しておく)

A＝(A＋2)MOD8

●2つの変数を使う

変数の値を切り換えるときに、別の変数を利用する方法もある。

例：変数Aの値を、変数Zを使って擬似的に放物線を描くように再現する

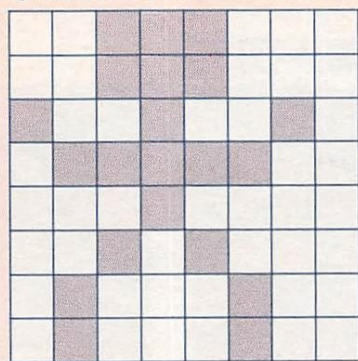
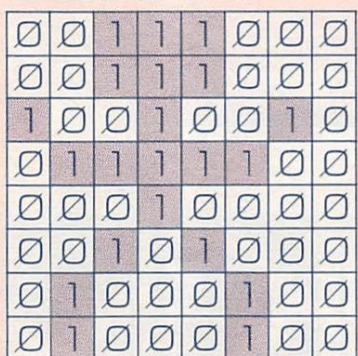
B＝B－1：A＝A＋B

●配列変数を使う

変数の値を切り換えるとっておきの手段が、配列変数を使って切り換える方法だ。この場合、配列変数と添字用の変数が必要。

例：変数Aを10種類の数値で循環させる(配列変数Zに10種類の数値を設定し、添字用変数にCを使う)

C＝(C＋1)MOD10：A＝Z(C)

■スプライトパターンデータの作り方

③-b 横1列ごとに、Øと1を2進数とみなして16進数にし、そのままつなげて文字列にする

```
&H38
&H38
&H92
&H7C
&H1Ø      3838927C1Ø284444
&H28
&H44
&H44
```

③-a 横1列ごとに、Øと1を2進数とみなして数値化する

```
&BØØ111ØØØ =  56
&BØØ111ØØØ =  56
&B1ØØ1ØØ1Ø = 146
&BØ11111ØØ = 124
&BØØØ1ØØØØ =  16
&BØØ1Ø1ØØØ =  40
&BØ1ØØØ1ØØ =  68
&BØ1ØØØ1ØØ =  68
```

④各列のデータをCHR$関数を使って文字に変換してつなげる

```
CHR$(56)  = "8"
CHR$(56)  = "8"
CHR$(146) = "い"
CHR$(124) = "¦"
CHR$(16)  = ※注参照
CHR$(40)  = "("
CHR$(68)  = "D"
CHR$(68)  = "D"
```

①スプライトパターンを作るときは、まず、どんなパターンを作るか元絵を作っておこう。スプライトは基本的に2色しか使えないので、単色で描いたほうがいいぞ。
②つぎに元絵の形のある部分、つまりドットにするところを1、そうでないところを0に置き換えていこう。形から数値に変換するための大切な作業なのだ。
③図では2つに分かれているが、まずaのように、元絵に振った0と1を2進数とみなして数値に変換する。よくわからなければ、横1列ぶん抜き出して、「PRINT　&B」と打ちこんだあとに付け加えて、リターンキーを押せば値がわかる。

このままでも十分パターンデータとして使えるが、状況によって2通りに分かれる。

まず、スプライトパターンをいくつも定義する場合や、データにコントロールコード(※)などを含むときは、③－bのように16進数にして、文字データとして使う。「PRINT　HEX$(データ)」とやれば、16進数に変換できる。このデータをもとに定義する方法は、次ページのサンプルを参照してほしい。

SPRITE$に直接定義するときは、数値をCHR$関数を使って文字に変換する。それらの文字をまとめてSPRITE$に設定すればOKだ。

※注意：0〜31、34、127、144、160、254、255のデータは、コントロールコード、文字列に使えない文字、打ちこめない文字になるので、これらのデータがあるときは、③－bのような方法を使うとよい

【スプライトの表示】スプライトはPUTSPRITEを使って表示される。書式は、「PUTSPRITEスプライト面番号，(X座標，Y座標)，色，スプライトパターン番号」となっていて、X座標とY座標はスプライトの表示位置、色は表示される色。まあ、わかるよね。問題なのは、スプライト面番号とスプライトパターン番号の2つで、スプライト面番号というのは、画面の手前のほうか奥のほうかということ。スプライトパターン番号はSPRITE$で定義した形の番号だ。この2つはまったく別のものなので、混同しないようにしよう。

プログラムに組み立てる

STICK入力とスプライトの切り換えを紹介したので、これらを使ったプログラムを組んでみよう。

■フローチャートを作ろう

プログラムを組むときは、かんたんなものでも図4のようなフローチャートを作るといい。

フローチャートはプログラムの設計図みたいなもので、これを作るということは、どう進行していくか、こんなときはどうするかを、あらかじめ考えておく作業をするということだ。

図4の場合でいうと、まず、スプライトパターンを定義したりする「初期設定」が始めにくる。

初期設定はどんなプログラムでも必ずある部分で、配列変数の宣言や画面モードの設定など、プログラムを実行していくうえでの環境を整える部分だ。

つぎにくる「スティック入力」は、今回のテーマのひとつであるSTICK関数を使って、入力を受け付ける部分だ。

そして、「座標計算」で入力値を座標増分に変換して、座標を計算しなおす部分だ。

「スプライト表示」は、座標が変われば、スプライトの表示位置も変更しなくてはいけないので、座標計算のすぐ下に位置しているのだ。また、このとき、スティック入力値により、もうひとつのテーマ、スプライトパターンの切り換えも行っている。

「トリガー入力判定」と「レンガの表示」はおまけで付けた。

さあ、それではプログラムを組んでみよう。

■サンプルの紹介

下のリストはフローチャートをもとに作成したサンプルだ。

カーソルキーの入力により、カーソルのスプライトが4方向移動する。そして、そのとき、カーソルのスプライトパターンも方向によって変化するのだ。スペースキーを押すと、その位置にレンガを表示する。この部分については、リストの解説と図5および図6を参考にしよう。

また、自分でオリジナルなもの、例えば、スペースキーを押すと音がなるとかに変更して、やってみてほしい。

次回のテーマはまだ未定だが、名前の入力なんてどうだろう。

(MORO)

●図4 かんたんなフローチャートを作る

画面モードや画面の色の設定、およびデータを読みこんでスプライトパターンの定義を行う。はじめに1度実行すればいい部分。

テーマのひとつ、STICK関数を使ったスティック入力の部分。入力値を変数に保存するのを忘れないようにしよう。

入力から座標増分に変換して、カーソルの座標を計算する部分。今回はかんたんな4方向の増分に変換することにした。

座標が変更されたら、スプライトの表示も更新しなくてはいけない。座標計算とセットにしておこう。また入力によりパターンを切り換えて表示している。

スペースキーによるトリガー入力判定。入力がなければスティック入力へ行き、あれば次に進む。ここでは連続入力を処理することにした。

スペースキーが押されたら、この処理が実行される。サンプルではレンガの文字をカーソルの位置に表示しているが、自分でオリジナルなものに変えてみよう。

●図5 VRAMアドレスの計算方法

●キャラクタパターンジェネレータテーブル

アドレス:[文字コード]×8の位置から8バイト、

設定するデータ:スプライトパターンと同様にデータを作り、数値のままVPOKEで書きこむ。文字コードはASC関数を使うと便利

●カラーテーブル

アドレス:&H2000+[文字コード]¥8

設定するデータ:文字の色×16+背景色の値を書きこむ。サンプルのように、16進数を使うとわかりやすい

●図6 IMPを使った連続入力防止法

T IMP Qの演算結果

Q＼T	0	−1
0	−1	0
−1	−1	−1

IMPは論理演算子のひとつで、その演算結果は左表のとおりだ。特定の組み合わせだけ0になるかわりものだ。

リストの行100のIF文では、IMPの演算結果が0のとき、入力があったと認めるわけだ。これが0になるは、今入力があり(T=−1)、前回はなかった(Q=0)ときだけなのだ。

■ひとつの結果としてのサンプル(ファイル名:SAMPLE.SB2)

```
10 SCREEN1,0:COLOR15,4,7:WIDTH32:KEYOFF:
DEFINTA-Z
20 FORI=0TO4:READA$:B$=""
30 FORJ=0TO7:A=VAL("&H"+MID$(A$,J*2+1,2)
):B$=B$+CHR$(A):NEXT:SPRITE$(I)=B$:NEXT
40 A=ASC("@")*8:READA$
50 FORI=0TO7:B=VAL("&H"+MID$(A$,I*2+1,2)
):VPOKEA+I,B:NEXT:VPOKE&H2008,&HF6
60 X=16:Y=8
70 'main
80 S=STICK(0):M=(S=7)-(S=3):N=(S=1)-(S=5
):X=(X+M+32)MOD32:Y=(Y+N+23)MOD23
90 PUTSPRITE0,(X*8,Y*8-1),15,2+M+N*2
100 Q=T:T=STRIG(0):IFTIMPQTHEN80
110 LOCATEX,Y:PRINT"@";:GOTO80
120 'sprite pattern data
130 DATA 183C66C381000000:'up
140 DATA 183060C0C0603018:'left
150 DATA FF818181818181FF:'no move
160 DATA 180C060303060C18:'right
170 DATA 00000081C3663C18:'down
180 'character pattern data
190 DATA 0404FF404040FF04
```

10 画面初期化/ファンクションキーの表示禁止/変数を整数型に宣言
20〜30 スプライトパターンデータ読みこみ/パターンデータの展開・定義
40〜50 キャラクタパターンデータ読みこみ/パターン定義/色設定
60 座標初期化
70 REM文
80 スティック入力/座標計算
90 スプライト表示
100 トリガー入力判定⇨入力がなければ行80へ
110 レンガの文字表示/行80へ
120 REM文
130〜170 スプライトパターンデータ(行20で読みこみ)
180 REM文
190 キャラクタパターンデータ(行40で読みこみ)

■実行画面

■キャラクタパターン

⬅レンガのパターンは「@」に定義されている。白い部分がドットのあるところで、赤い部分は背景の部分だ

■スプライトパターン

0 1 2 3 4

SUPER BEGINNERS'講座

スーパー ビギナーズ

超初心者

第3回

今月のテーマ：文字列の入力処理

ゲームに登場する主人公に自分で名前を付けられるのは、より楽しくするための大切なくふうだろう。今回は名前入力に代表される、文字列の入力処理を紹介しよう。

文字を入力する

むかし、ファンダムで作品の評価をしていたとき、こんなものがあった。

「名前が変更できます。変更のしかたは、リストの×行と×行のPRINTの後の○○○を、好きな名前に変えてください」。

確かにそうすれば、どんな名前にも自由に変更できるけど、できればリストを改造しなくてもすむようにしたいものだ。

そこで今回は、文字の入力について紹介したいと思う。

■MSXの文字

文字入力の前に、MSXの文字の種類を覚えておこう。

ちょっと表1を見てほしい。もう見飽きた、という人もいるかもしれないが、これはMSXの画面に表示される文字を集めたもので、キャラクタコード表と呼ばれている。

そして、左側にちょっと離れて並んでいるのがグラフィックキャラクタで、GRAPHキーを押しながら打ちこむ文字だ。

■文字の入力

ところで、画面にカーソルが出ているときに文字キーを押すと、画面にその文字がすぐ表示されるけど、カーソルが画面に出ていないとき、例えばとても長いプログラムのリストを表示しているときや、何かの命令を実行しているときに文字キーを押しても、すぐにはその文字が表示されない。このようなときに入力された文字は、カーソルが次に現れたときに画面に表示されるのだ。

よく考えると、ちょっと不思議なことなんだけど、あまり気にしたことはないかな？

MSXでは、キーバッファと呼ばれるところに、キーボードから打ちこまれた文字を保存しているのだ。だからプログラムを実行しているときやリストを表示しているときでも、文字の入力が受け付けられているのだ。

そして必要に応じて、そこから文字が送り出されてくるのだ。

右の実験1でやっていることを、自分のMSXでちょっと試してみて欲しい。

■プログラムの文字入力

プログラムでキーボードから文字の入力を受け付ける場合、このキーバッファから文字を拾ってくることになる。だから実験1のように、入力命令が実行される前にこのキーバッファに文字が入っているときは、その文字からプログラムに渡されてしまうのだ。

キーバッファには文字の情報以外にも、RETURNキーやCLSキーなどが押されたという情報も入るので、状況にもよるが、入力命令が実行される前に、キーバッファの中をカラにしておく必要があるのだ。

■表1 キャラクタコード表

	4	5	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		π		0	@	P	`	p	♠			ー	タ	ミ	た	み
1	月	┴	!	1	A	Q	a	q	♥	あ	。	ア	チ	ム	ち	む
2	火	┬	"	2	B	R	b	r	♣	い	「	イ	ツ	メ	つ	め
3	水	┤	#	3	C	S	c	s	♦	う	」	ウ	テ	モ	て	も
4	木	├	$	4	D	T	d	t	○	え	、	エ	ト	ヤ	と	や
5	金	┼	%	5	E	U	e	u	●	お	・	オ	ナ	ユ	な	ゆ
6	土	│	&	6	F	V	f	v	を	か	ヲ	カ	ニ	ヨ	に	よ
7	日	─	'	7	G	W	g	w	ぁ	き	ァ	キ	ヌ	ラ	ぬ	ら
8	年	┌	(	8	H	X	h	x	ぃ	く	ィ	ク	ネ	リ	ね	り
9	円	┐	)	9	I	Y	i	y	ぅ	け	ゥ	ケ	ノ	ル	の	る
A	時	└	*	:	J	Z	j	z	ぇ	こ	ェ	コ	ハ	レ	は	れ
B	分	┘	+	;	K	[	k	{	ぉ	さ	ォ	サ	ヒ	ロ	ひ	ろ
C	秒	╳	,	<	L	¥	l	¦	ゃ	し	ャ	シ	フ	ワ	ふ	わ
D	百	大	-	=	M	]	m	}	ゅ	す	ュ	ス	ヘ	ン	へ	ん
E	千	中	.	>	N	^	n	~	ょ	せ	ョ	セ	ホ	゛	ほ	
F	万	小	/	?	O	_	o		っ	そ	ッ	ソ	マ	゜	ま	■

■実験1 キーバッファを体験しよう

```
FORI=0TO1:I=-STRIG(0):NEXT:INPUT A$■
```

↑写真のように打ちこもう。次にRETURNキーを押して実行すると、カーソルが消える

↑キーボードから適当に文字を入力してみよう。ただし、まだスペースキーは押さないこと。ここでは入力したはずの文字が画面に表示されないことを確認しておこう

```
FORI=0TO1:I=-STRIG(0):NEXT:INPUT A$
? EFGLOKFJGLASOE087120.+-//*:;:;トイスリキニクテ
イ ■
```

↑最後にスペースキーを押すと、今まで入力した文字がドバッと画面に現れる。キーバッファには40文字までしか保存されないので、画面に現れた文字の数も40文字までになる

●キャラクタコード表の赤い文字は16進数で、それぞれの対応する文字の番号、つまりキャラクタコードを表している。上の16進数が上位の桁を表し、左の16進数は下位の桁を表す。例えば、"A"のキャラクタコードなら&H41（10進数で65）になる。●グラフィックキャラクタはグラフィックヘッダと呼ばれるコントロールコードの文字が付くので、画面では1文字でも、文字列としては2文字ぶんの大きさになる。例えば、A$="月"とした場合、変数A$の内容は、CHR$(1)+CHR$(&H41)となっているのだ。

名前入力の処理を作る

プログラムで文字の入力を受け付けるには、表2にあるような命令が用意されている。

このうち、もっとも手軽に使えるのがＩＮＰＵＴで、これを使うと数値入力も可能だ。

ただ手軽なぶん、ＩＮＰＵＴでは受け付けることができない文字があったり、入力にミスがあると、メッセージが出て改行してしまうので、画面構成が崩れてしまったりする。

なんとなく安っぽい感じがするからだろうか、完成度の高い力作ではあまり使われない。

しかし、１画面などの短い作品でちょっとした工夫をするのには、最適な命令ではある。

■入力処理を組む

入力処理ではどのようなことをすればいいのだろう。

まず入力を受け付ける前に、キーバッファをカラにする必要があるだろう。これはほとんどの場合に必要な処理だ。

加えて、１～３のどれかを選んで欲しいとき、０や10、－７などの範囲外の入力がそのまま通ってしまい、プログラムの動作に影響があっては困る。

数値とは限らず、入力された文字を表示するときに、文字数が多くて画面が崩れても困る。

ただ入力を受け付けるだけではなく、入力が条件にあっているかを判定する必要がある。

■表２　今月登場した命令などの一覧

ＩＮＰＵＴ"メッセージ"；変数
実行するとメッセージと「?」を表示して入力待ちの状態になり、リターンキーが押されるまでに入力された文字または数値が変数に入る。数値型の変数なら数値以外は受け付けず、文字型の変数の場合はコントロールコードや「"」、「,」以外の文字を受け付ける。
メッセージを省略して、
ＩＮＰＵＴ　Ａ＄
のように使用することもできる。また、
ＩＮＰＵＴ　Ａ＄，Ｘ，Ｂ＄
のように、「,」で区切って複数の入力を受け付けることもできる。

文字変数＝ＩＮＫＥＹ＄
キーバッファから指定された文字変数に１文字設定する。ＣＴＲＬ＋Ｃキーまたは ＣＴＲＬ＋ＳＴＯＰキー以外のすべての文字を入力できるが、この命令を実行してもカーソルは表示されず、入力待ちにもならない。

文字変数＝ＩＮＰＵＴ＄(文字数)
指定された文字数ぶんの入力があるまで待ち、文字変数に設定する。入力できる文字はＩＮＫＥＹ＄とおなじだが、ＢＳキーなどの入力は、文字として加えられえるだけなので、いちど入力した文字は修正することができない。

■実験２　文字コードを調べる方法

```
PRINT ASC(INPUT$(1))
```

写真のように打ちこんで、リターンキーを押して実行しよう

```
PRINT ASC(INPUT$(1))
 8
Ok
```

調べたい文字を入力すると、その文字コードが表示される。写真はＢＳキーを押した場合だ

他にも、ＩＮＰＵＴ以外の入力命令を使う場合、入力している文字が画面に表示されないものや、カーソルさえも画面に表示されないものがある。だから、きちんと入力できているか目で見て確認できるように、画面に表示する必要がある。

またこれらの命令では、ＢＳキーやＤＥＬキーも入力文字として扱われてしまうので、修正することができない。できればプログラムで修正ができるようにくふうしたほうがいいだろう。

プログラムでＢＳキーなどのコントロールキャラクタを判別するには、上の実験２のようにして、それらの文字コードを調べておくとよい。実際の判定や処理については下のサンプルを参考にして欲しい。

このように、入力処理はただ受け付けるだけではなく、いろいろと配慮しなければいけない部分がある。

また、使用する命令によって、入力を受け付ける部分にくふうが必要な場合もあるのだ。

```
input name >ホレーショーくん  <

input message >キミは まだ゛なにもしらない <
ホレーショーくん

キミは まだ゛なにもしらない
Ok
```

サンプルリストを実行したときの画面だ。このサンプルでは、グラフィックキャラクタは入力できないが、ちょっとくふうすればグラフィックキャラクタも入力できるようになる

■完成した名前入力処理

```
10 DEFINTA-Z:SCREEN1:WIDTH32:COLOR15,4
20 M$="name":Y=3:L=10:GOSUB100
30 NM$=A$
40 M$="message":Y=6:L=15:GOSUB100
50 ME$=A$
60 PRINT:PRINT NM$:PRINT:PRINT ME$
70 END
100 'name sub ========================
110 A=0:A$="":GOSUB200:GOTO170
120 I$=INKEY$:IFI$=""THEN120
130 IFI$=CHR$(13)THENRETURN
140 IFI$=CHR$(8)ANDA>0THEN180
150 IFI$<" "ORI$=CHR$(127)THEN120
160 IFA=LTHEN120 ELSEA$=A$+I$:A=A+1
170 LOCATEX,Y:PRINTUSING"input "+M$+" >&
"+SPACE$(L-2)+"&<";A$:GOTO120
180 A=A-1:A$=LEFT$(A$,A):GOTO170
200 'key buffer clear sub =============
210 IFINKEY$=""THENRETURNELSE210
```

左のリストの行100～180の部分が文字列入力サブでＩＮＫＥＹ＄を使って入力を受け付けている。パラメータ用の変数を設定してＧＯＳＵＢで呼び出せば、Ａ＄に入力された文字列が入ってもどってくるのだ。

●変数の意味

【文字列入力サブのパラメータ用】
Ｍ＄……入力受け付け時に表示されるメッセージ用
Ｘ、Ｙ……入力文字およびメッセージの表示座標
Ｌ……入力を受け付ける文字数
【その他の変数】
Ａ……現在入力された文字数
Ａ＄……現在入力された文字列
Ｉ＄……キー入力用
ＮＭ＄、ＭＥ＄……入力された文字列の保存用

●プログラム解説

【メイン部】
10　変数の型宣言／画面モード設定／１行の文字数設定／画面の色設定
20　メッセージ、表示位置、入力文字数設定／文字列入力サブ呼び出し
30　入力された文字列の保存
40　メッセージ、表示位置、入力文字数設定／文字列入力サブ呼び出し
50　入力された文字列の保存
60　保存しておいた文字列の表示
70　プログラムの終了
【文字列入力サブ】
100　ＲＥＭ文
110　入力用変数初期化／キーバッファクリアサブ呼び出し／行170へ
120　キー入力受け付け／入力がなければこの行を繰り返す
130　入力終了判定⇒リターンキーが入力されたらもとの処理にもどる
140　修正判定⇒ＢＳキーの入力があり、文字数が１以上なら行180へ
150　入力の範囲外判定⇒コントロールコードなら行120へ
160　入力文字数判定⇒入力文字数に達していれば行120へ／そうでなければ入力用変数更新
170　表示位置設定／メッセージ、入力文字列表示／行120へ
180　入力用変数更新／行170へ
【キーバッファクリアサブ】
200　ＲＥＭ文
210　キー入力受け付け／入力がなければもとの処理へ／あればこの行を繰り返す

●INPUT、INKEY$、INPUT$の他に、LINEINPUTという入力命令もある。使い方はINPUTに似ているが、指定できる変数は文字型の変数のみで、複数指定することはできない。しかし、「,」や「"」も入力できるので、場合によっては便利だ。●文字列入力サブのサンプルプログラムは、都合により今月の付録ディスクに収録できなかったので、グラフィックキャラクタも入力できるようにした、汎用文字列入力サブを来月の付録ディスクに収録しようと思う。

超初心者

SUPER BEGINNERS'講座

スーパー ビギナーズ

第4回

今月のテーマ ループ処理とは

プログラムを組む上で、基本的となる処理のひとつがループ処理だ。ループ処理にもいろいろあるが、今回はFOR～NEXTの基本的な部分について紹介する。

繰り返しおこなう処理

プログラムで「A」という文字を画面に5つ、横に並べて表示するにはどうすればいいだろうか。

答えはかんたんで、

10 PRINT "AAAAA"

と、始めから「A」を5つ並て表示するようにすればいいのだ。

では、縦に5つ並べて表示するにはどうすればいいだろうか。

カーソルキーとAキーを使って画面に並べるなんていうのはダメ。どんなプログラムにすればいいか、ちょっと考えてみてほしい。

■「A」を縦に5つ表示するプログラム

では、答えを見てみよう。下にあるリストがこの問題の答えだ。

「ぜんぜんわからなかった」という人は、リスト1のプログラムを見てほしい。

行番号を付けて、

PRINT "A"

を5つ並べただけの、PRINT文さえ知っていれば、かんたんに組むことができるプログラムだ。

「かんたんだった」という人の中には、リスト1のようなプログラムができた人もいるのではないだろうか。

では次に、リスト1の下に掲載されている、リスト2のプログラムを見てほしい。このプログラムでも、「A」という文字を縦に5つ表示することができるのだ。

すでに気が付いただろうけど、リスト1にはPRINT文が5つあったが、リスト2では1つしかないのだ。かわりに始めの行に、

FOR I=1 TO 5

というのがあり、終わりには、

NEXT

というのがある。

これは2つで1セットになっていて、あいだにあるPRINT文を5回繰り返すという意味の命令だ。

つまりリスト1より短いプログラムで、「A」を縦に5つ並べて表示できるということになる。

そしてこれが、今回のテーマになっている**ループ処理**を実現する、FOR～NEXT（フォアー～ネクスト）ループという。

ループ処理とは、このように繰り返しておこなう処理のことだ。

■リスト1 「A」を縦に5つ表示するプログラム

```
10 PRINT "A"
20 PRINT "A"
30 PRINT "A"
40 PRINT "A"
50 PRINT "A"
```

■リスト2 FOR～NEXTを使って表示する

```
10 FOR I=1 TO 5
20 PRINT "A"
30 NEXT
```

■写真1 リスト2の実行画面

```
10 FOR I=1 TO 5
20 PRINT "A"
30 NEXT

run
A
A
A
A
A
Ok
█
```

↑このように「A」が縦に5つ表示され、実行結果はリスト1と変わらない

PRINT文

PRINT文はメッセージや式の値を表示するための命令だ。書式は

PRINT　表示データ

のようになり、表示データには文字列でも、数式でも、それらを組み合わせたものでも指定可能だ。表示される位置は、カーソルのある位置からになるので、LOCATE文(カーソル位置指定の命令)と組み合わせて使われることが多い。表示後は改行されてカーソル位置が次行の左端に移動するが、データの最後に「；」(セミコロン)を付けた場合には、改行されずに表示の末尾にカーソルが移動する。

ループ処理

今回はFOR～NEXTを使用したループ処理のみを紹介しているが、他にもたくさんのやり方がある。例えば、リスト2をIF文とカウンタ用の変数を使って、

```
10　A=0
20　PRINT "A"
30　A=A+1
40　IF A<5 THEN 20
```

のようにしておこなうこともできる。他にも、ON　GOTO文を使ってループ処理を組んだり、GOTO文を使った無限ループなどもある。ファンダムの1画面プログラムの解説などを参考にして、調べてみよう。

FOR〜NEXTの基礎知識

FOR〜NEXTループは、FOR〜の部分とNEXTの部分にわかれている。それぞれの部分について説明するので、右の図や下のリストを参考に読んでほしい。

■FOR〜NEXTの仕組み

FOR〜の部分は、ループを繰り返す回数などを定義する部分になっていて、以下の3つのことを指定しなければいけない。

①そのループで使う**カウンタ用の数値変数**(ループ変数)を指定する
②指定されたループ変数の始めの値を指定する
③指定されたループ変数の値が、いくつになるまでループを繰り返すか、終値を指定する

これら3つのことを、

FOR ①=② TO ③

というぐあいに指定してやると、ループが定義されるのだ。

次にNEXTの部分だが、これはループを実行する部分になる。

具体的に説明すると、FOR〜の部分で指定されたループ変数の値を1増やす。その結果、ループ変数の値が終値以下ならば、FOR〜の直後の処理にもどる。もし終値を超えていれば、そこでループを終了して次の処理に進むのだ。

■ループ変数の値とSTEP

FOR〜の部分では、第4の指定としてSTEPというのがある。

これはNEXTの部分でループ変数に加算される値を指定するものだ。だから、下の実験2や実験3のように指定すれば、ループ変数を奇数で変化させたり、減らしていったりすることができるのだ。省略することができ、その場合は1ずつ加算されることになる。

FOR〜NEXTループでは、ループ変数がとても大事な役割をはたしているということが、わかってもらえただろうか。

今回は紹介できなかったこともあるので、次回も引き続きループ処理を紹介したいと思う。

■実験1 リスト2のループ変数

```
FOR I=1 TO 5:PRINT I;:NEXT
 1  2  3  4  5
Ok
█
```

↑リスト2では、ループ変数の値が1ずつ増加していき、5回繰り返しているのがわかる

■実験2 2ずつ増えていくタイプ

```
FOR I=-1 TO 4 STEP 2:PRINT I;:NEXT
-1  1  3
Ok
█
```

↑STEP2を指定すると、ループ変数は2ずつ増える。結果的に3回繰り返すループになる

■実験3 1ずつ減っていくタイプ

```
FOR I=2 TO -2 STEP -1:PRINT I;:NEXT
 2  1  0 -1 -2
Ok
█
```

↑このように、STEPでマイナスの値を指定すると、ループ変数の値もだんだん減っていく

■図1 FOR〜NEXTの使い方

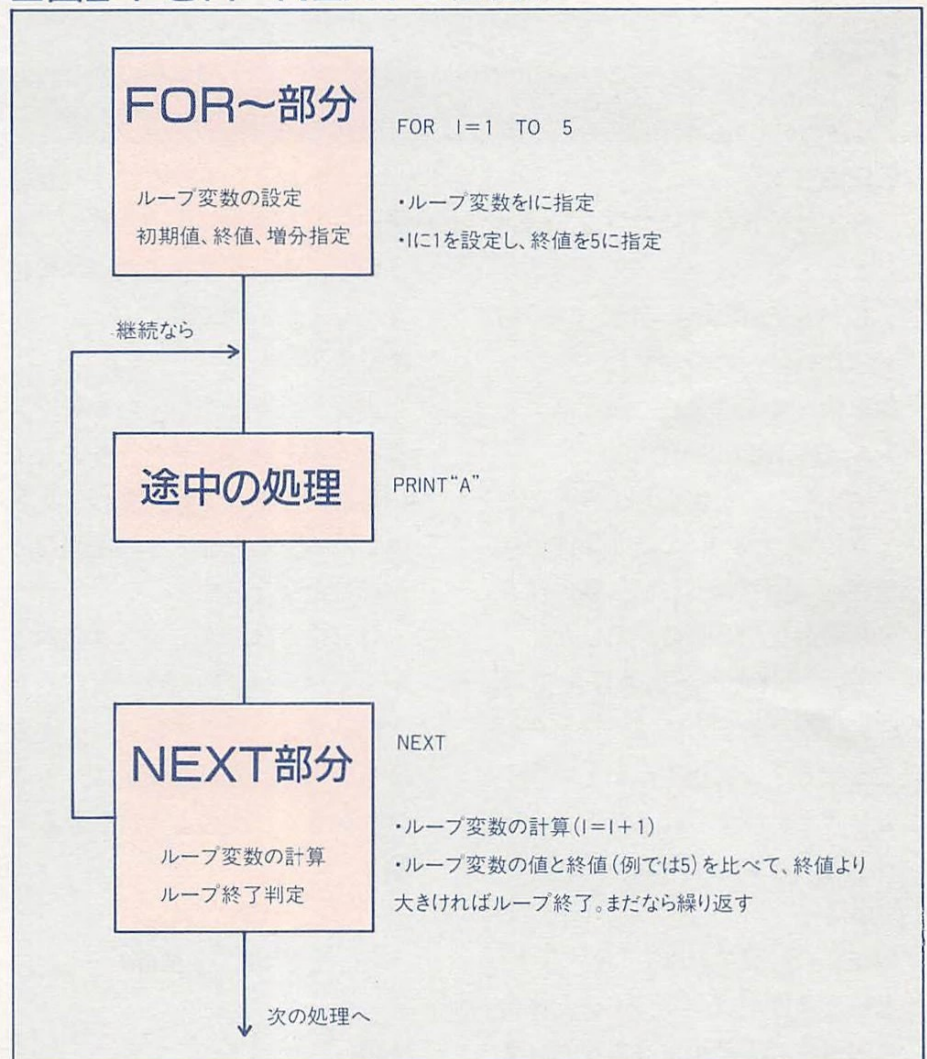
FOR【数値変数】=【初期値】TO【終値】

NEXT

【数値変数】 ループカウンタ、ループ変数という。ここで指定する変数の条件は、数値変数であること。注意することは、ループ中指定した変数をもとにループが実行されるので、ループ中はむやみに値を壊さないようにすること。

【初期値】 ループ変数の始めの値を指定する。

【終値】 ループ変数の終わりの値を指定する。ループはループ変数の値がこの指定の値を超えると終了する。

■図2 FOR〜NEXTの仕組み

■応用編 ここでリターンキーを押すと?

```
FOR I=1.5 TO 3.9:PRINT I;:NEXT█
```

```
FOR I%=1.5 TO 3.9:PRINT I%;:NEXT█
```

↑実行するとわかるが、2つの結果は異なるのだ。相違点は下のループ変数に%という記号がついていること。これは変数の型を表す記号で、整数型を意味する。詳しくは下の解説で

プログラムのご相談・修理・仕立て MORO'Sショップ開店

ファンダムに送られてくる質問ハガキの中にはプログラミングに関する質問も多い。SB講座やBASICピクニックの記事で扱ったり、かんたんなものならスクラムの質問箱で答えたりしてきたが、このカコミと付録ディスクで答えることにした。

☆どう組むか、どう直せばいいか、どうすればうまくいくかといった、プログラムに関するご相談・修理・仕立ての注文を受け付けます。「SB講座MORO店」まで送ってください。なお、料金はいりません。

カウンタ用の変数
カウンタ用の変数とは、カウント、つまり数えるための変数という意味で、ループ処理で使われる場合には、そのループ回数、つまり処理を繰り返した回数を数えるための変数だ。FOR〜NEXTループでは、FOR文でこの変数を指定する必要があるので、特にループ変数と呼んでいる。

【応用編の解説】
2つの相違点である%という記号は、整数型を表す記号なのだ。初期状態では変数はすべて実数型なので、上の場合には問題なく動作するが、下の場合にはループ変数にこの記号が付いて整数型になっているので、FOR部分で始めの値を設定するときに小数部分が切り捨てられ、I%が1の状態でループが実行されてしまう。そのため実行結果が異なってしまうのだ。

SUPER BEGINNERS' 講座

超初心者

スーパー ビギナーズ

第5回

今月のテーマ
いろいろなループ処理

前回はFOR～NEXTループの基本を紹介したが、今回は、2つのループを組み合わせた2重ループなど、よく使われる、いろいろなループ処理を紹介する。

前回は、

プログラムで、「A」という文字を縦に5つ並べて表示するにはどうすればよいか

という問題から、**2つのサンプルリスト**を紹介した。

始めのものは、

PRINT "A"

を5つ並べてあるだけの、単純なプログラムだった。

そして次に紹介したリストでは、このPRINT文が1つしかなく、代わりに始めの行に、

FOR I=1 TO 5

というのがあり、終わりには、

NEXT

というのがある、たった3行からなるプログラムだった。

この2つめのプログラムで使われている、FOR～とNEXTという2つのものが、ループ処理を実現するものであった。

■ループ処理の利点

始めのリストは見た目に結果がわかりやすいが、例えば表示する文字を、「A」から「B」に変えようとしたとき、5つのPRINT文すべて変更しなくてはいけない。

しかしループ処理を使ったものでは、1つのPRINT文を変更すれば済んでしまう。

ループ処理の利点は、おなじ作業を繰り返して行うとき、リストが節約でき、かつ変更が容易になるということになる。

FOR～NEXTループの基本は前回紹介したので、ちょっと応用を紹介しようと思う。

■ループの中にループ

では、リスト1のプログラムを見てほしい。行10は画面を初期化している部分で、画面モードと色、表示文字数を指定している。

そして、行20にはFOR～文があるのだが、次の行を見るとまたFOR～文があるのだ。

これは間違いではなく、ループを2重に定義しているのだ。行20では**ループ変数**にYが指定されているが、行30ではXになっている。

ここが重要なポイントだ。

この2重ループが、どのように実行されるのか追ってみよう。

行20でYのループ、行30でXのループがそれぞれ定義され、行40が実行される。

行50のNEXTは、最後に定義されたFOR～文、つまり、Xのループに対応している。だから、Xのループが繰り返されるのだ。

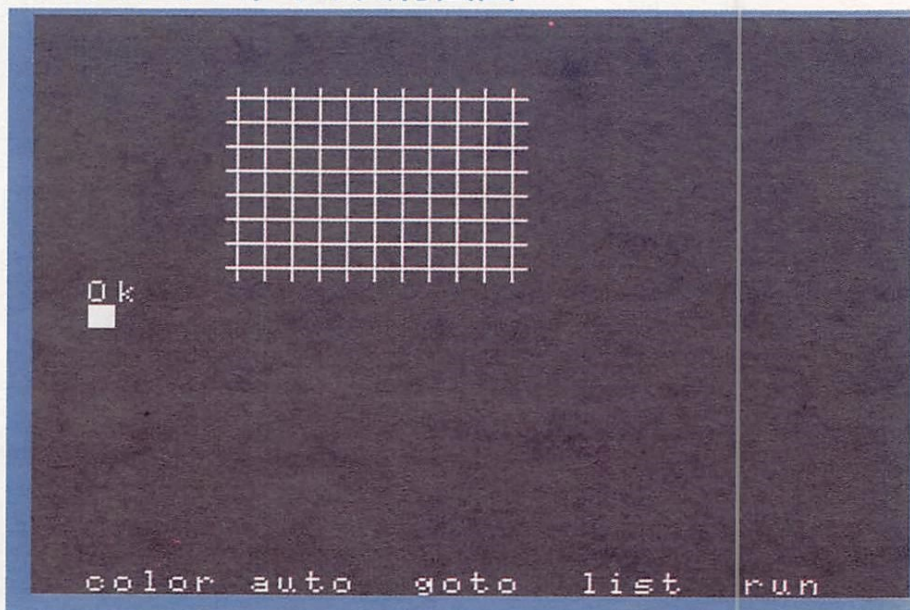
ではXのループが終了したあとはどうか。行60のNEXTでは、Xのループは終了しているので、その前に定義されたYのループが繰り返されるのだ。

Yのループのもどり先は行30のXのループを定義する部分になる。

つまり、この2重ループでは、Yのループの回数だけXのループを繰り返す、ということになる。

■リスト1 文字を四角形に並べる

```
10 SCREEN 1:COLOR 15,4,7:WIDTH 29
20 FOR Y=3 TO 10
30  FOR X=5 TO 15
40   LOCATE X,Y:PRINT "+"
50  NEXT
60 NEXT
```

←行40の最後から2文字目はGRAPH+Fで入力します

■写真1 リスト1の実行画面

↑リスト1の実行画面。グラフィックキャラクタを2つのループで縦と横に並べて表示する

2つのサンプルリスト
前回掲載したリストとは、
10 PRINT "A"
20 PRINT "A"
30 PRINT "A"
40 PRINT "A"
50 PRINT "A"
と、
10 FOR I=1 TO 5
20 PRINT "A"
30 NEXT
の2つのプログラムです。

ループ変数
FOR～NEXTループでは、ループ変数と呼ばれる変数を使ってループを管理しているので、FOR～文で定義するときに、必ず指定する必要がある。

行40のLOCATE文
リスト1の行40にあるLOCATE文では、ループ変数のXとYを使って、文字を表示する位置を指定している。Xが5、Yが3のとき、
LOCATE X,Y
は、テキスト座標の(5, 3)の位置にカーソルが移動することになる。
テキスト座標とは、画面のいちばん左上を(0, 0)とし、X座標は右へ何文字目かを表し、Y座標は下へ何文字目かを表すものだ。

似たような処理
例えば、配列変数の値をまとめて設定する場合、FOR～NEXTループがよく使われる。2つ3つなら、個別に設定してもそれほどではないが、数が多くなると大変だからだ。
また、グラフの計算などでもループが使われる。1つ1つの点の計算を連続して行える利点があるからだ。
FOR～NEXTループは、ほとんどのプログラムに使われている、とても利用価値のある命令なのだ。

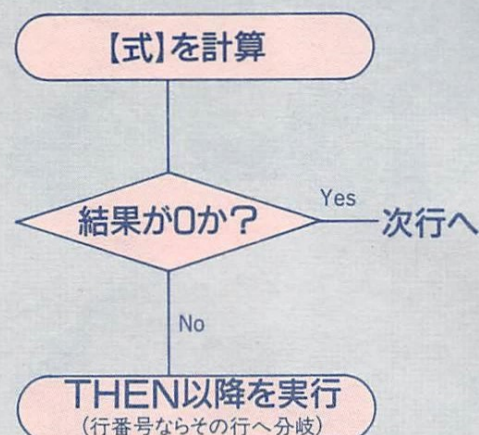
■図1 IF～THEN文

(a)IF～THEN文の使い方

IF 【式】THEN 【文】

または、【行番号】

●式の結果が0以下なら、THEN以降の文を実行、または、行番号で指定された行へ分岐する
●式の結果が0なら、次行へ移行する
●式は数式、関係演算、論理演算など、結果として数値が得られるもの。定数や、文字列に対する関係演算も指定可能

(b)IF～THEN文の仕組み

IF～THEN文は、以下のような仕組みで実行されるのだ。
①式を評価する
IF～THEN文に与えられた式を計算・評価し、結果として数値を得る
②結果を判定して分岐する
結果が0でなければTHEN以降へ分岐し、0ならば次行(もしくはELSE以降)へ分岐する
③ELSE以降の実行
ELSE以降に文があれば実行する。また、行番号があればその行へ分岐する

リスト1のプログラムは、2重ループのほかに、もうひとつの意味が含まれている。

それは、**行40のLOCATE文**を見るとわかるが、ループ変数の値を利用しているということだ。

つまり、ループ変数の値を利用することで、ただ同じ処理を繰り返すだけでなく、**似たような処理**をひとつにまとめて、連続して実行することができるのだ。

FOR～NEXTループでは、ループ変数の値を利用できることが最大の魅力なのだ。

■待つためのループ処理

ここまで紹介してきたループ処理は、繰り返すことが目的だった。つまり、繰り返す回数が始めから決まっているわけだ。

しかし、ループ処理はそれだけではないのだ。

例えばゲームなどで、タイトル画面が出てから、スペースキーが押されるまでそのまま待っている場合がある。

じつはこれもループ処理の目的のひとつで、待つためのループ、つまり、繰り返す回数が決まっていないループなわけだ。

右の写真2にあるPLAY文を、ちょっと実行してみてほしい。

音が鳴ればメロディはどうでもいいのだが、音が鳴っているのに、「Ok」とカーソルが表示されたのがわかっただろうか。

PLAY文で音を鳴らすとき、鳴っている最中でも何か別のことができるのだ。そのため、例えばタイトルでBGMを鳴らしたら、ゲームが始まったのにタイトルの音楽が鳴っている、なんてことになったりする。

こんなとき、ループが使われる。

■リスト4 3秒間待つ(その1)

```
10 TIME=0
20 FOR I=0 TO 180:I=TIME:NEXT
```

■リスト5 3秒間待つ(その2)

```
10 TIME=0
20 IF TIME < 180 THEN 20
```

■写真2 音を鳴らす

```
PLAY "V15O4L4 CDEFGAB O5C"
Ok
█
```

■リスト2 音が鳴り終わるまで待つ(その1)

```
10 PLAY "V15O4L4 CDEFGAB O5C"
20 FOR I=-1 TO 0:I=PLAY(0):NEXT
```

■リスト3 音が鳴り終わるまで待つ(その2)

```
10 PLAY "V15O4L4 CDEFGAB O5C"
20 IF PLAY(0) THEN 20
```

リスト2の行20にあるループは、PLAY文が鳴り終わるのを待つためのループ処理だ。

PLAY(Ø)というのは、音が鳴っていれば－1、鳴っていなければ0という値になる関数だ。

そしてその関数の値が、なんとループ変数に代入されているのだ。

このループの終値は0なので、Iが0のときNEXTまでいけば終了するのだが、音が鳴っているあいだはIが－1にされてしまうので、ループが終わらないのだ。逆に音が鳴り終わると、Iは0になり、ループが終了するのだ。

もうひとつ例を見せよう。

リスト4は3秒間待つループで、TIMEは60分の1秒ごとに＋1される変数だ。ループの終値が180なので、TIMEが180つまり3秒たたないとループが終了しない。

このように、ループ変数の値を変更することで、ループの回数を調節することができるのだ。

■IF文を使ったループ

リスト3とリスト5は、これらのリストとおなじ目的の処理で、IF文を使ったループ処理だ。

待つループの場合IF文を使うほうがスマートになる。IF文については図1を参考にしてほしい。

PLAY(Ø)
これは、PLAY関数というもので、引数に対応するチャンネルの演奏状態を返す関数だ。この関数が使用されたとき、演奏中であれば－1、そうでなければ0を値とする。引数との対応は、
0＝全チャンネル(1～3のどれか1つでも演奏中なら－1になる)
1＝Aチャンネル
2＝Bチャンネル
3＝Cチャンネル
となっている。

TIME
これは、タイマー変数というもので、60分の1秒ごとに、自動的に値が＋1されていくシステム変数だ。変数なので、値を変更することができるので、時間を計る場合などで使われる。また、プログラムが実行されているかどうかに関係なく、値が更新されているので、よく乱数初期化に使われたりする。
整数型なため、0～65535の範囲で値が繰り返される。

プログラムのご相談・修理・仕立て MORO'Sショップ

プログラミングに関する質問は、ハガキや封書はもちろんのこと、ディスクを付けての質問も受け付けている。ただし、その場合には、ディスクは返送できないので、じゅうぶん注意してほしい。また、勝手ではあるが、テープは遠慮してほしい。
☆どう組むか、どう直せばいいか、どうすればうまくいくかといった、プログラムに関するご相談・修理・仕立ての注文を受け付けます。「SB講座MORO店」まで送ってください。なお、料金はいりません。

※質問のお答えは誌面を通じて行う予定です。

SUPER BEGINNERS' 講座

スーパー ビギナーズ

超初心者

ループ処理に続いて、知っておきたい基本的な処理がＩＦ～ＴＨＥＮ文を使った条件分岐だ。状況に応じて分岐する処理は、プログラムには欠かせない存在だ。

前回のループ処理でもちょっと紹介したＩＦ～ＴＨＥＮ文だが、これはもともと条件分岐命令だ。

ある条件、前回の例でいえば、音が鳴っているかどうかという条件を判断し、その結果、プログラムを分岐するものだった。

前回紹介した例では、その分岐の結果がループ処理になっているだけのことで、ＦＯＲ～ＮＥＸＴ文のように、ループのための命令とは違うものだ。

今回は、このＩＦ～ＴＨＥＮ文(以下ＩＦ文)の一般的な使われ方である条件分岐について紹介する。

ＩF~THEN~ELSE

まずＩＦ文の動作をかんたんに説明しよう。

条件となる式が成り立つかどうかを判定して、成り立つ場合にはＴＨＥＮ文へ、成り立たない場合にはＥＬＳＥ文へ移行する。

ＴＨＥＮ文では、行番号が指定されている場合にはその行の先頭にジャンプする。そうでない場合には、そこにある命令などを実行する。その後ＥＬＳＥ文を見つけたら、そこでその行の実行を終了して次行へジャンプする。

ＥＬＳＥ文もＴＨＥＮ文の場合とおなじように実行されていく。

このようにＩＦ文は実行されるのだが、いまいちピンとこないという人のために、リスト１のプログラムを例に確認してみよう。

リスト１のプログラムを走らせると、画面に、

A=?■

と表示されるので、適当な数値を入力してリターンキーを押そう。すると、いま行20のＩＦ文で実行されたのがＴＨＥＮ文かＥＬＳＥ文か画面に表示されるのだ。

行20のＩＦ文には、

A<5

という条件式が与えられている。つまりこのＩＦ文では、入力された数値(変数Ａ)が、５より小さいかどうかを判定しているのだ。

この判定により、ＴＨＥＮ文が実行された場合には、

M$="THEN"

が実行され、ＥＬＳＥ文が実行された場合には、

M$="ELSE"

が実行されるのだ。

そしてＭ＄に設定された文字列を行50で表示しているので、いまどちらが実行されたかわかるのだ。

■リスト1 THENかELSEか

```
10 INPUT "A=";A
20 IF A<5 THEN M$="THEN" ELSE M$="ELSE"
30 PRINT "A=";A;"ﾉ ﾄｷ､"
40 PRINT "ｺﾉ IF ﾌﾞﾝ ﾊ､";
50 PRINT M$;" ｦ ｼﾞｯｺｳ ｼﾏｼﾀ"
60 PRINT
70 GOTO 10
```

■写真1 リスト1の実行画面

```
list
10 INPUT "A=";A
20 IF A<5 THEN M$="THEN" ELSE M$="ELSE"
30 PRINT "A=";A;"ﾉ ﾄｷ､"
40 PRINT "ｺﾉ IF ﾌﾞﾝ ﾊ､";
50 PRINT M$;" ｦ ｼﾞｯｺｳ ｼﾏｼﾀ"
60 PRINT
70 GOTO 10
Ok
run
A=? 0
A= 0 ﾉ ﾄｷ､
ｺﾉ IF ﾌﾞﾝ ﾊ､THEN ｦ ｼﾞｯｺｳ ｼﾏｼﾀ

A=? 5
A= 5 ﾉ ﾄｷ､
ｺﾉ IF ﾌﾞﾝ ﾊ､ELSE ｦ ｼﾞｯｺｳ ｼﾏｼﾀ

A=? -1
A=-1 ﾉ ﾄｷ､
ｺﾉ IF ﾌﾞﾝ ﾊ､THEN ｦ ｼﾞｯｺｳ ｼﾏｼﾀ

A=? ■
```

注意してほしいのは、ＴＨＥＮ文が実行されて、Ｍ＄に文字列が代入されたあとのところだ。

次にあるのがＥＬＳＥ文なので、ここでＴＨＥＮ文の実行が終了し、次行へジャンプしているのだ。

行20のＩＦ文の条件式を変えていろいろ試してみよう。

ＩF~GOTO~ELSE

あまり大したことではないが、ＩＦ文では、ＴＨＥＮの代わりにＧＯＴＯを使うこともできる。

この場合の動作は、ＴＨＥＮを使ったときとまったく変わらない。ただ見た目が違うだけだ。だから、リスト１の行20のＩＦ文を、

20 IF A<5 GOTO M$="GOTO"~

ＩNPUT
ＩＮＰＵＴ文は指定した変数に入力を受け付ける命令だ。このとき指定した変数の型によって入力を受け付ける。だから、リスト１のように数値変数が指定されているときに数値以外の文字を入力すると、
？Ｒｅｄｏ　ｆｒｏｍ　ｓｔａｒｔ
とメッセージが出て、もう一度入力をやり直すことになるので注意。

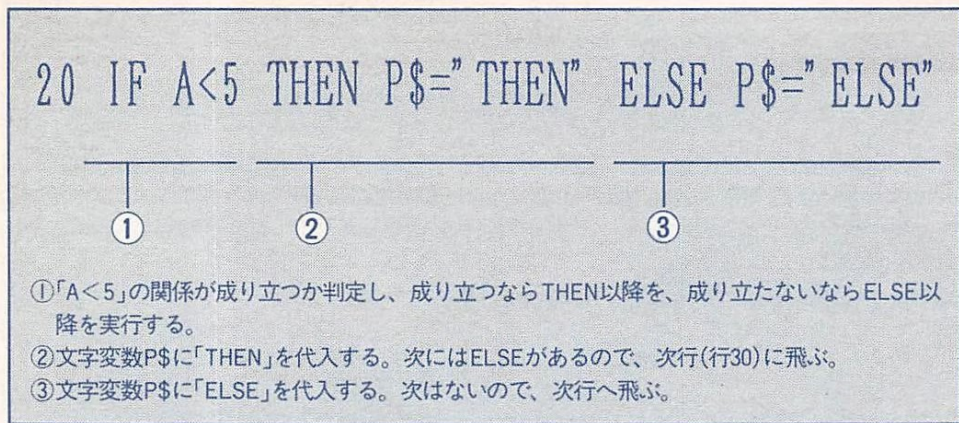
ＩF~GOTO
今回、始めのうちＩＦ～ＴＨＥＮ文とこだわっていたのは、このＩＦ～ＧＯＴＯ文があるためだ。どちらもおなじ動作をするので、とくに覚える必要はないのだが、投稿作品に使われていることがあるので紹介した。行番号へのジャンプのときはＧＯＴＯにするなど、区別する意味では使えるかもしれない。しかし、ふつうのＧＯＴＯ文と混同しやすいので注意が必要だ。

GOTO
指定した行の先頭にジャンプする命令。ＧＯＴＯ文の後に何が書かれていても、その部分は無視される。たまにそれを利用して、コメントが書いてある投稿作品がある。

STRIG(0)
スペースキーが押されていれば－１を、押されていなければ０を返してくれる。また、カッコの中の数値によって、
１＝ポート①のトリガーＡ
２＝ポート②のトリガーＡ
３＝ポート①のトリガーＢ
４＝ポート②のトリガーＢ
が押されているかどうかを調べることができる。

■図1　リスト1のIF文の動作

```
20 IF A<5 THEN P$="THEN" ELSE P$="ELSE"
```

①「A＜5」の関係が成り立つか判定し、成り立つならTHEN以降を、成り立たないならELSE以降を実行する。
②文字変数P$に「THEN」を代入する。次にはELSEがあるので、次行(行30)に飛ぶ。
③文字変数P$に「ELSE」を代入する。次はないので、次行へ飛ぶ。

■リスト2 スペースキーでBEEPを鳴らす（その1）

```
10 IF STRIG(0)=0 THEN 10
20 BEEP
30 GOTO 10
```

■リスト3 スペースキーでBEEPを鳴らす（その2）

```
10 IF STRIG(0)=0 THEN 10
20 BEEP
30 IF STRIG(0) THEN 30 ELSE 10
```

というふうに、行20を書き換えてもまったく問題なく動くのだ。

ちょっと不思議な感じもするが、これも立派なＩＦ文なのだ。

ＩＦ文による分岐

リスト２はスペースキーを押すとビープ音が鳴るプログラムだ。実行してみるとわかるが、押しているあいだはずっと鳴っている。そしてリスト３はスペースキーを押した回数だけ、ビープ音が鳴るプログラムだ。

２つのプログラムの相違点は、行30にある。リスト２ではたんにＧＯＴＯ文で行10にジャンプしているだけだが、リスト３ではＩＦ文を使ってスペースキーを放すまでループするようになっている。そしてスペースキーが放されれば、ＥＬＳＥ文により行10へジャンプするのだ。このような処理を連続入力防止処理という。

行30のIF文のTHEN以降を、
THEN 20 ELSE END
に変えて実行してみよう。

スペースキーを押すとビープ音が鳴り出し、離すとプログラムが終了するだろう。

リスト１の例では、一定の流れにそってプログラムが実行されていたが、このようにＩＦ文を使うことで状況によってプログラムの流れを変えることができるのだ。

ＩＦ文は、うまく使えば効率のよいプログラムを組むことができるのだ。しかし逆にいえば、ＩＦ文を使い過ぎると、プログラムの流れが複雑になり過ぎて自分にも把握(はあく)できなくなったり、かえって効率の悪いプログラムになったりしてしまうのだ。

ところで、行130のIF文は関数だけで「条件式」になっている。

ＩＦ文では与えられた式の値が０ならＥＬＳＥ文を、０以外ならＴＨＥＮ文を実行するのだ。

ＳＴＲＩＧ(0)という関数は、スペースキーが押されているときは－１、押されていなければ０を返すので、－１ならＴＨＥＮが、０ならＥＬＳＥが実行されるのだ。

連続入力防止処理
例えば、『テトリス』ではスペースキーでブロックを回転させるが、押しているあいだずっと回りっぱなしだったらとても困る。このように、１回の入力が何回分もの入力とされないようにするのが、連続入力防止処理なわけだ。ＳＴＲＩＧ関数やＳＴＩＣＫ関数など、回数で区切れない入力方法を使うときに施されることが多い。

END
プログラムを終了する命令。ＧＯＴＯ文のように、ＥＮＤの後に何が書かれていても無視される。

プログラムのご相談・修理・仕立て

MORO'Sショップ

【質問】いま英単語学習ツールを作成しているのですが、単語の新規登録、検索、登録してある単語の中からランダムに選んで出題するレッスン機能を備えるつもりですが、単語の新規登録のところで壁にあたっています。単語の意味を入力するのに漢字を使用したいので、漢字モードのＳＣＲＥＥＮ５で作っていますが、ＩＮＰＵＴ文で入力するとＳＣＲＥＥＮ１にもどってしまいます。ＩＮＫＥＹ＄で入力すると、漢字で入力できません。ＳＣＲＥＥＮ５でプログラムを作るのが正しいのか悪いのかもわかりません。間違っていると思われる部分を指摘してください。また、役に立つテクニックなんかも教えてください。

（長野県・阿部健一／17歳）

記念すべき第１号のお客様は、長野県の阿部健一くんと決まりました。おめでとう、……ん？なんか違うなあ、ありがとう!?でもないし……。まあ、いいか。

手っとり早くいうと、英単語ツールを作ろうとしたら、いきなり入力のところでつまずいた、ということのようですな。

■INKEY$も漢字OK

阿部くんは試行の末、漢字を入力するにはテキスト画面でないといけないのだろうかという疑問を抱いたようですが、そんなことはありません。

漢字をプログラムで扱うときのポイントとして、漢字１文字はＭＳＸのキャラクタ２文字に相当するということ。だから、ＩＮＫＥＹ＄で入力させる場合、２文字分入力があって、始めて漢字１文字の入力になるのです。

下に掲載してあるリストは、グラフィック画面での漢字入力サブのサンプルプログラムです。付録ディスクに、
ＭＯＲＯ'Ｓ．ＳＢ８
というファイル名で収録してあるので、組みこんで試してみてください。

このプログラムでは、入力にＩＮＫＥＹ＄を使っていますが、それほど変わったことはやっていません。ただＢＳキーの入力があった場合に１文字削除するようにしたので、
＿ＫＬＥＮ()
＿ＫＭＩＤ()
という２つの漢字モード専用の拡張命令を使っています。

これらはＬＥＮやＭＩＤ＄といった関数とおなじ働きがあり、漢字、つまり全角文字もふつうの半角文字も、どちらも１文字として扱ってくれます。使い方などはマニュアルを調べてみてください。

また、阿部くんの場合には、わざわざ処理を組まなくても、入力のときはＳＣＲＥＥＮ１でＩＮＰＵＴ文を使うほうが手っ取り早いかもしれません。

１つのプログラムを作成する過程には、例えば検索やデータの保存、読みこみといったところで、いくつも障害がある場合があります。そういったときは今回の阿部くんのように、まずいろいろ試してみることが大切です。そういった苦労があれば、それに見合うだけの力が付いてくるものです。もし、それでもだめだったときには、当店まで気軽にご相談ください。

☆プログラムに関するご質問・修理・仕立てなどのご注文は、「ＳＢ講座ＭＯＲＯ店」へどうぞ。なお、料金はいりません。

■漢字入力サブのサンプル　（ファイル名：MORO'S.SB8）

```
1 'save"A:moro's.sb8":'
10 _KANJI:CLEAR 200:DEFINT A-Z
20 KEYOFF:COLOR 15,4,7:SCREEN 5:CLS
30 X=0:Y=2:GOSUB310:END
300 '===================== input sub ==
310 A$="":GOSUB 410:GOTO 370
320 I$=INKEY$:IF I$="" THEN320
330 IF I$=CHR$(13) THEN RETURN
340 IF I$=CHR$(8) THEN 380
350 IF I$<" " OR I$=CHR$(127) THEN 320
360 A$=A$+I$
370 LOCATE X,Y:PRINT A$"_   ":GOTO 320
380 _KLEN(A,A$):IF A=0 THEN 320
390 _KMID(A$,A$,1,A-1):GOTO 370
400 '========== key buffer clear sub ==
410 IF INKEY$="" THEN RETURN ELSE 410
```

SUPER BEGINNERS' 講座

超初心者

スーパー ビギナーズ

第7回

今回のテーマ
プログラムの分岐

プログラムを分岐させるには前回のIF文でも行える。しかし、分岐先がたくさんあるときには使いづらい。今回はそういった分岐が得意な命令を紹介する。

前回紹介したIF文では、条件を指定して、成立するかどうかで2つの分岐が起こった。その条件が成り立つときはTHEN以降の部分を実行し、成り立たなければELSE以降の部分を実行する。IF文を使うことで、1つの行を2つにわけていたわけだ。

しかし、もしこういった分岐を同時に3つ以上行いたいときにはどうすればよいだろうか。

IF文を使って3つ以上の分岐を行おうとした場合、複数のIF文を使わなければ実現できない。IF文を並べて、ひとつずつ判定しているのならまだ見やすいが、ファンダムの掲載作品などによく見かける、**複合IF文**なんてものになると、とてもわかりにくく、よくバグの原因になったりもする。

今回はそういった、たくさんの分岐をいちどに行うための命令を紹介する。

ON~GOTO文

前回、IF文とまったくおなじ動作をするIF~GOTO文というのを紹介したが、これとは違い、**ON~GOTO文**は分岐のための命令なのだ。

なんとなく、違いに気付かないかもしれないが、この命令での分岐は「条件」によるものではない。

リスト1を見てほしい。行10の**INPUT文**により変数Aに入力された数値が入る。次の行20にはON~GOTO文があり、「~」の部分に変数Aが置かれ、GOTOのあとには100、200、300とカンマで区切られた数値が並んでいる。

カンのいい人ならわかったかもしれないが、変数Aの値によって、GOTOの後にある行番号へジャンプするのだ。

具体的には、変数Aが1のとき1番目の行番号にジャンプする。Aが2なら2番目、3なら3番目となるのだ(画面1)。

このとき、もしAの値が0や4だったらどうだろうか。この場合、Aに対応する行番号が指定されていないので、ON~GOTO文は無視される(画面2と3)。

このようにON~GOTO文を使えば、3つ以上の分岐をとてもわかりやすい形で実現できるのだ。

ON~GOTO文を使うときはAがマイナスの値にならないように注意しよう。マイナスの値だとエラーが出てしまうのだ(画面4)。

■リスト1　ON~GOTO文のサンプル

```
10 INPUT "number";A
20 ON A GOTO 100,200,300
30 PRINT "ﾑｼ ｻﾚﾏｼﾀ."
40 END
100 PRINT "one :";A
110 END
200 PRINT "two :";A
210 END
300 PRINT "three :";A
310 END
```

■画面1　1を入力した場合

```
run
number? 1
one : 1
Ok
█
```

■画面2　4を入力した場合

```
run
number? 4
ﾑｼ ｻﾚﾏｼﾀ.
Ok
█
```

■画面3　0を入力した場合

```
run
number? 0
ﾑｼ ｻﾚﾏｼﾀ.
Ok
█
```

■画面4　−1を入力した場合

```
run
number? -1
Illegal function call in 20
Ok
█
```

複合IF文
複合IF文とは、IF文のあとにまたIF文がある形をいう。下のリストは複合IF文の形になっていて、まず、始めのIF文により、変数Aが0より大きいかどうかで分岐が起こる。もしAが0より大きいのであれば、直後のTHEN部分が実行されるが、そこにはさらにIF文があるので、また分岐が起こる。もしAが0未満なら、始めのIF文でELSE部分が実行されるわけだが、1つめのELSEは2つめのIF文に対応しているので、2つめのELSE部分ということになる。複合IF文が多いプログラムは、このように動作がわかりにくく、混乱のもとになる。

```
10 IF A>0 THEN IF B=0 THEN C=1 ELSE C=0
ELSE B=1:A=ABS(A)
```

ON~GOTO
ON~GOTO文の「~」の部分には、結果として0以上の数値を返すものならば、計算式でも関数でもかまわない。ただし、定数では意味がないので注意。また、分岐先の行番号はきちんと指定する必要があるが、例えば、
ON A%*2 GOTO,100,,200,,300
のように、その数値に必ず偏りが起き、選択されることがあり得ないような行番号の部分を省略することはできる。

INPUT文
INPUT文は指定した変数に入力を受け付ける命令だ。このとき指定した変数の型によって入力を受け付ける。そのためリスト1のように数値変数が指定されているときに、数値以外の入力を行うと、
?Redo from start
というメッセージが出て、適切な入力がされるまで、やり直すことになる。

■画面5　1.9を入力した場合

```
run
number? 1.9
one  :  1.9
Ok
■
```

もうひとつ注意してほしいことがある。それは画面5のように、ON～GOTO文に小数を指定した場合だ。

ON～GOTO文では、つねに指定された数値の整数部分だけを見ている。そのため画面5のように1.9を入力した場合、1を入力したときとおなじになるのだ。

次に、小数が入力されたときも違いがある。ON～GOTOでは整数部分によって分岐が起きたが、リスト2では無視される（画面7）。

IF文では、条件式が成り立つかどうかで分岐するのに対して、ON～GOTO文では登録された行番号のうち、何番目の行にジャンプするかで分岐するのだ。

ＩＦ文とON～GOTO文

ちょっとリスト2を見てほしい。これは冒頭で述べた、ＩＦ文を使った複数の分岐例だ。リスト1と動作が似ているが、大きな違いが2つある。ただし、行番号の違いや行へのジャンプになっていないことは除外してのことだ。

まずマイナスの値が入力された場合だ。リスト1のように、ON～GOTO文ではエラーが出るが、リスト2ではマイナスの値が入力されてもエラーが出ない（画面6）。

ON～GOTO文とGOTO文

ON～GOTO文は分岐のための命令だ。しかし、指定によって飛び先が変わるものの、基本的にはGOTO文とおなじくジャンプする命令には違いない。飛び先がたくさんあるか、ひとつしかないかの違いくらいなものだろう。

では右上にあるリスト3と4を見てほしい。どちらも元はおなじプログラムだったが、リスト3はGOTO文のあとに、リスト4はON～GOTO文のあとに、それぞれ「ERROR!」とラクガキがしてある。

■リスト3　GOTO文のうしろ

```
10 A%=RND(1)*2
20 PRINT A%
30 ON 1-STRIG(0) GOTO 30
40 GOTO 10:ERROR!ERROR!ERROR!!!
```

■リスト4　ON～GOTO文のうしろ

```
10 A%=RND(1)*2
20 PRINT A%
30 ON 1-STRIG(0) GOTO 30:ERROR!!!!!!
40 GOTO 10
```

REM文になっているわけではないので、どちらもエラーが出そうだが、片方は何事もなく動作し、エラーが出るのはひとつだけだ。

結論からいうと、GOTO文が実行されると指定した行にジャンプしてしまうので、通常はあとに何が書かれていても無視される。

しかし、ON～GOTO文の場合には、実行されたときに、必ずジャンプするわけではないので、エラーが出ることがあるのだ。

ジャンプが起きない場合というのは、「～」の部分に0や行番号の個数よりも大きい数値が指定されたときのことだ。

このプログラムでは、ON～GOTO文の「～」の部分に、
1－STRIG(0)
が指定されている。STRIG関数はスペースキーが押されていないときは0を返すので結果は1になる。そして1番目に指定されている行30にジャンプする。つまりこの行を繰り返すのだ。

スペースキーが押されるとSTRIG関数は－1を返す。1から－1を引く、つまり1＋1で2になり、対応する行番号がないため無視されて次の文が実行される。このときリスト4ではラクガキが実行されてエラーが出るのだ。

2つのリストを実行して試してみるといい。　(MORO)

■リスト2　IF文を使ったサンプル

```
10 INPUT "number";A
20 IF A=1 THEN PRINT "one :";A:END
30 IF A=2 THEN PRINT "two :";A:END
40 IF A=3 THEN PRINT "three :";A:END
50 PRINT "ﾑｼ ｻﾚﾏｼﾀ."
60 END
```

■画面6　－1を入力した場合

```
run
number? -1
ﾑｼ ｻﾚﾏｼﾀ.
Ok
■
```

■画面7　1.9を入力した場合

```
run
number? 1.9
ﾑｼ ｻﾚﾏｼﾀ.
Ok
■
```

ERROR
じつはこれもBASICの命令の１つで、わざとエラーを出すための命令だ。この命令の正しい書式は、
ERROR <エラーコード>
なので、リスト4では本物のエラーが出るようになっている。
まあしかし、故意にエラーを出すような状況は、めったにないので、覚える必要も使う必要もないのだが。

REM文
プログラムの中にコメントを書くときなどに使う。MSXはREM文を見つけると、そこでその行の実行をやめて次の行を実行する。REM文の多くはプログラムのまとまりを示すのに使用されている。私はよくやっているが、よく使うちょっとした命令の集まりをREM文にしてリストに入れておけば、けっこう利用価値がある。もちろん、たんなるラクガキをしてもいいのだ。

プログラムのご相談・修理・仕立て　MORO'Sショップ

■表計算ソフトの依頼
☆依頼☆職場で100名くらいの給料の合計と金種を計算する仕事があるのですが、そういった計算のできるプログラムを載せていただけないでしょうか。
（東京都・江端敬一／？歳）

まず始めのお客様は、表計算プログラムを掲載して欲しいとのことですが、本格的なものはとても掲載できません。しかし、限られた内容のものであれば、サンプルとして作り、掲載することもできます。ただちょっと、これだけの情報では作れません。計算の具体的な内容や項目数といった情報が必要ですし……。そういった情報が届き次第、検討してみます。

■グラフィックの変換法
☆質問☆私は障害児の教育に、A1STを利用して学習を実践している者です。そこで教えていただきたいことがあります。SCREEN8で作成した画像をSCREEN5に変換する方法を教えていただきたい。
（和歌山県・松本光央／51歳）

こちらのお客様はたくさんのハガキを送ってくれた、熱心なMSXユーザーの方です。本当のことをいいますが、私はグラフィック関係と音楽関係は苦手でして、ただいま編集部Orcに方法を聞いているところで、次号でお答えする予定です。

■1から教えてください
☆質問☆RPGやSLGなどが作れるようになるには、まず、どんなことを覚えていけばよいのでしょうか。
（神奈川県・伊藤一智／16歳）

3人目のお客様は長文だったので要約させていただきました。人生相談に近い内容でしたが、残念ながらプロになるためのアドバイスはできません。私自身、趣味の域から抜け出していないからです。しかしプログラムを覚えていく上でのアドバイスはあります。始めからプロなどと考えず楽しんで組むこと。これだけです。教えられるより自分から学ぶほうが身に付きます。まず覚えることは、BASICやマシン語の命令でしょう。

SUPER BEGINNERS' 講座

スーパービギナーズ

超初心者

第8回
今回のテーマ
サブルーチン処理

プログラムには流れがある。その流れを制御するのがループ処理や条件分岐といったものなのだ。そして今回は、いよいよサブルーチンの登場だ。

プログラムの流れ

プログラムというのは、命令が実行される順番に並んでいるのがふつうだ。

しかし、ただ実行される順番に並べておくだけでは、やたら長くなったり、状況によっての特別な処理を作ることができない。

そこでループ処理や条件分岐といった命令を使ってプログラムの流れを制御することで、このような不満を解消するのだ。

似たような処理が続くときにはFOR～NEXTのようなループ処理を使うことで、それらをひとつの処理にまとめることができる。状況によって特別な処理を行いたいならIF文やON～GOTOを使って分岐させればいい。

サブルーチン

似たような処理が続く場合にはループ処理でじゅうぶんなのだが、もし、似たような処理がたくさんあるのに、続いていないときにはどうすればいいだろう。

例えば、ちょっと凝った効果音を鳴らすPLAY文が、いくつかの場所で使われている場合などはいい例だろう。

プログラムの中で何度も使われているからといって、これはさすがにループ処理にはできない。

それぞれのPLAY文の間には、何らかの処理があるためだ。しかしこのままではプログラムが長くなるし、MMLを変更したときに直し忘れて不都合が生じる可能性だってある。

では、PLAY文を単独の行に置き、演奏したいときにその行に飛ぶようにしたらどうだろうか。この場合、鳴らすまではうまくいくのだが、鳴らし終わったときにどこへもどればいいかが問題になってくる。もどり先は複数存在するからだ。

変数とON～GOTOを使ってもどるという方法もある。しかしこれでは、もどり先がつねに行の先頭になってしまい、演奏場所を変えたときや演奏か所を増減したときに、修正がかなり面倒になる。ちょっと前振りが長くなったが、このような場合は**サブルーチンを使う**と、とてもきれいにまとめることができるのだ。

サブルーチンでは、もどり先の指定をする必要がなく、呼び出し元の次の命令にもどるのだ。

■リスト1 サブルーチン処理のサンプル

```
10 CLS
20 PRINT "ｷｮｳ ﾉ ﾃﾝｷ ﾊ･･･"
30 P$="(1.ﾊﾚ 2.ｸﾓﾘ 3.ｱﾒ 4.ﾕｷ)"
40 MX=4:GOSUB 110:WE=A
50 PRINT "ｲﾏ ﾉ ｼﾞｶﾝ ﾊ･･･"
60 P$="(1.ｱｻ 2.ﾋﾙ 3.ﾖﾙ)"
70 MX=3:GOSUB 110:TM=A
80 END
90 :
100 'ｻﾌﾞﾙｰﾁﾝ
110 A=0:PRINT P$;:INPUT A
120 IF A<1 OR A>MX THEN 110
130 PRINT:RETURN
```

GOSUBとRETURN

サブルーチン処理を実現させるのが、GOSUBとRETURNという2つの命令だ。

GOSUBの働きはGOTOとよく似ている。基本的には指定した行に飛ぶ命令だが、そのとき、いまどこにいるかといった情報を**記憶してから**指定した行に飛ぶ。つまり、あらかじめもどってくる場所を記憶してからサブルーチンに飛ぶ命令なのだ。

RETURNはサブルーチンの処理が終わったときに使う命令で、実行するとGOSUBで記憶した場所にもどっていく。このとき、**誤動作をしないように**、記憶したもどり先を忘れる。

つまり、何度も呼び出す処理の終わりにRETURNと書いておくだけでサブルーチンにできる。そしてその処理を、

GOSUB 〈行番号〉

と実行して呼び出せば、どこから実行しても、きちんと元の場所にもどってくるのだ。

サブルーチンの例

リスト1はサブルーチン処理のサンプルで、数値の入力と判定の処理を行100～130をまとめてサブルーチンにしている。

このサブルーチンは行40と行70にある、

GOSUB 110

で呼び出されて実行されるのだ。

MML
Music Macro Languageの略。PLAY文で音楽を演奏するときに使用する文字で、A～Gの文字で音の高さを表し、Oで音階、Lで音の長さ、Vで音量、Tでテンポを表すなど、音楽を演奏するための働きを持った文字。実際にPLAY文で使用するときは、数値を付加したりして、これらの文字をつなげて演奏することが多い。

サブルーチンを使う
サブルーチンはある程度の長さのプログラムを組むときには必要な手段だ。プログラムの節約になるのはもちろんのこと、細かい処理をサブルーチン化すれば、メイン部分を簡潔に組めるので、全体が把握しやすいプログラムにできる。またミスがあった場合には、どこをチェックすればいいか、わかりやすいという利点もある。

記憶してから
実際にGOSUBで記憶されるのは、GOSUBの次にある命令の位置情報が記憶される。これはプログラムが実行されたとき、MSXはプログラムカウンタというもので次に実行する命令の位置を示している。このプログラムカウンタの内容がGOSUBによって記憶されるからだ。また参考までに、情報を記憶すると、メモリを7バイト消費するので覚えておこう。

誤動作をしないように
GOSUBでもどり先の情報を記憶すると、メモリが7バイト消費される。もしサブルーチンからRETURNでもどったときに、記憶した情報が消えなかったら、サブルーチンを呼び出すたびに7バイトずつメモリが消費され、いつかメモリが足りなくなってエラーが出る可能性がある。もとの処理にもどった時点で役目は終わるのだから、むだのないように消去されるのだ。

■リスト2 指定行にもどるサンプル

```
10 CLS
20 PRINT "ｷｮｳ ﾉ ﾃﾝｷ ﾊ･･･"
30 P$="(1.ﾊﾚ 2.ｸﾓﾘ 3.ｱﾒ 4.ﾕｷ)"
40 MX=4:GOSUB 110:WE=A
50 PRINT "ｲﾏ ﾉ ｼﾞｶﾝ ﾊ･･･"
60 P$="(1.ｱｻ 2.ﾋﾙ 3.ﾖﾙ)"
70 MX=3:GOSUB 110:TM=A
80 END
90 :
100 'ｻﾌﾞﾙｰﾁﾝ
110 A=0:PRINT P$;:INPUT A
120 IF A=0 THEN BEEP:RETURN 10
130 IF A<1 OR A>MX THEN 110
140 PRINT:RETURN
```

このサブルーチンの動作を見てみると、まず行110で入力用の**変数Aを初期化**し、メッセージを表示してINPUTを実行する。

行120では入力された変数Aの値が範囲内にあるかどうか判定し、範囲外であれば行110の入力をやり直すようになっている。

入力が範囲内であれば行130で空行を表示して、RETURNにより元の処理にもどるのだ。

変数を利用する

このサンプルでは、変数を引数として使うことで、サブルーチン処理の適用範囲を拡げている。

このサブルーチンで使っている変数は全部で3つあり、入力用の変数Aとメッセージ表示用の変数P$、そして、入力値の上限用の変数MXがある。

GOSUBでサブルーチン処理を呼び出す前に、変数P$にどのような数値を入力すればよいかのメッセージを設定し、変数MXには入力値の上限を設定することで、任意の範囲の数値を入力するサブルーチン処理にしているのだ。

そして入力された値は変数Aに残るので、RETURNでもどったときには、入力値は変数Aによって**参照できる**のだ。

先程の効果音の例の場合では、おなじ音楽を鳴らすということでサブルーチン処理を紹介したが、いくつか効果音を使う場合には、変数によってどの音楽を演奏するか指定するようにすれば、効果音をひとつのサブルーチンにまとめることも可能になる。

指定行にもどる

リスト2は、サブルーチン処理にやり直しの判定を入れた場合のサンプルだ。

プログラムを始めからやり直したいとき、もとの処理にもどってしまっては都合が悪いこともある。そこでこのサンプルでは、入力値が0の場合、行10にもどるようになっている。

RETURN (行番号)

のようにすると、もどり先の記憶を消した上で、指定した行に飛ぶことができるのだ。

こんなことをしなくても、たんにGOTO文を使えばいいように思えるが、もどり先の記憶を残しておくと、メモリを消費する上に誤動作の原因になるからだ。

また、もどり先の記憶がない状態でRETURNを実行すると、RETURN without GOSUB in ××の**エラーが出る**ので注意が必要だ。

プログラムのご相談・修理・仕立て MORO'Sショップ

前号で保留していた質問に、Orcから回答がありましたので、質問とともに紹介します。

■グラフィックの変換法

☆質問☆私は障害児の教育に、A1STを利用して学習を実践している者です。そこで教えていただきたいことがあります。SCREEN8で作成した画像をSCREEN5に変換する方法を教えていただきたい。

(和歌山県・松本光央/51歳)

SCREEN8のグラフィックをSCREEN5へ変換するには、まずそれぞれのVRAM(グラフィックのデータが格納されているRAMのこと)の構造を知る必要があります。

●VRAMの構造の違い

SCREEN8では1バイトで1ドットを表します。この内訳は、1ビットごとに

GGGRRRBB

となっていて、G(緑)とR(赤)は3ビットで0〜7の8段階、B(青)は2ビットで0〜3の4段階で色を指定できます。全部で256色の中から1ドットごとに色を指定できるのです。

つぎにSCREEN5ですが、VRAMに直接色が格納されてはいません。SCREEN5ではパレットという手法が使われ、VRAMにはそのパレットの番号が格納されています。

パレットは全部で16種類あり、それぞれについてRGB各8段階ずつ指定できるので、全部で512色の中から任意に選べることになります。パレットを使うことで1ドットの情報は4ビットで表せます。そのためVRAMの1バイトには2ドットぶんの情報が格納されています。

SCREEN8とSCREEN5の大きな違いは、このVRAMの使われ方にあり、SCREEN8では256色すべてが使えるのに対して、SCREEN5では512色中の16色しか使えないのです。

●変換についての考え方

元となるSCREEN8のグラフィックの色数が16色以下なら話は単純です。それぞれのパレットにおなじ色を設定して変換するだけで済むからです。

しかし色数が16を超えているときは、SCREEN5では16色までしか使えないので、各パレットにどんな色を設定するかを決めなければいけません。

パレットの色を決めるには、グラフィックツールなどで自分で適当な色を作ってもいいでしょう。また、元の絵を調べて、使われているドット数が多い色を優先してパレットに設定するのもいいでしょう。

●サンプルプログラム

付録ディスクにはサンプルプログラムを収録してあります。ファイル名:HENKAN. ASC このサンプルでは、SCREEN8の元絵に使われている色を調べ、て、使用頻度の多い色とRGBの距離が遠い色をパレットに設定するようにしてあります。

実行するときは行120にあるFR$とFW$にファイル名を設定してから実行してください。FR$が元絵のSCREEN8のグラフィック、FW$が作成するSCREEN5のグラフィックとなっています。またPAの値はパレット設定時のRGBの距離です。これを0にすると距離を無視し、大きくすると色がばらつきます。10〜20くらいが適当でしょう。

なお、このサンプルでの変換には恐ろしく時間がかかります。プログラムを高速化してターボRの高速モードで実行しても2時間弱かかります。実用的な速度に高速化するには、マシン語にする必要があるでしょう。

変数Aを初期化
INPUT文で入力を受け付ける場合、入力がリターンキーだけだったときは変数の内容がそのままになる。例えば、INPUT文の直前で、変数Aの内容が1だったとしよう。もしこのとき、リターンキーのみの入力であれば、変数Aの内容は変化せずに、1のまま次の処理に進んでしまうのだ。サンプルではこれを防ぐために、変数Aの値を0に初期化しているのだ。

参照できる
サブルーチンでなんらかの処理が実行され、その値が一時的に参照されるのでなければ、適切な変数に値を保存することを忘れないようにしよう。もどり値用の変数は、サブルーチンが実行されると内容が変わってしまう可能性があるからだ。サンプルでは変数Aの内容を、変数WEやTMに保存している。

エラーが出る
GOSUBによってもどり先の情報が記憶されていないときにRETURNが実行されるとエラーが出る。これはRETURNでもどることができないために出るエラーなのだ。サンプルでは行10からプログラムを実行していき、行80のENDでプログラムが終了する。ENDの後にはサブルーチンがあり、もしここでENDによってプログラムを終了させなかったら、もどり先の情報がない状態でサブルーチンが実行されてしまい、エラーが出てしまうのだ。サブルーチンを使う場合、このようにきちんと区別しておく必要があるのだ。

SUPER BEGINNERS' 講座

スーパービギナーズ

超初心者

第9回
今回のテーマ
割り込みサブ

サブルーチンにはちょっと変わった親戚がいる。GOSUBではなく、イベントという条件で呼び出されるものだ。今回はこの親戚を紹介しよう。

サブルーチンの役割

前回は似たような処理を1つにまとめるのに、サブルーチン処理が有効だという話をした。しかしサブルーチンはこれだけではない。

たとえばドラクエタイプのRPGを考えたとき、マップを表示する部分や戦闘場面などは、独立した処理であることが多い。

プログラムが長く大きなものになるほど全体の把握が難しくなる。そこで、プログラムのメイン部分を簡素化し、特定の状況の処理はそれぞれサブルーチンとして作成することで、大きなプログラムを組むことが容易になる。先の例でいえば、マップ表示や戦闘処理はそれぞれサブルーチンにしておき、必要なときに呼び出すようにしておくのだ。

大げさにいうと、1つのリストの中に複数のプログラムが同居しているようなもので、その橋渡しをしているのがGOSUBとRETURNというわけだ。

ON～GOSUB

ジャンプ命令にGOTOとON～GOTOがあったように、サブルーチンを呼び出す命令にもGOSUBとON～GOSUBがある。

ON～GOSUBはON～GOTOとおなじように、～の部分の条件によって、どのサブルーチンを呼び出すかが決まるものだ。

もちろんサブルーチンを呼び出す命令には違いないので、RETURNが実行されれば、ちゃんともとの場所にもどってくる。

割り込み処理

サブルーチンの親戚に割り込み処理というのがある。これはある状況になったとき、サブルーチンが自動的に呼び出されるものだ。

ある状況というのは、たとえば、トリガーボタンが押されたとか、スプライトが重なったなどのイベントのことを示す。

これらのイベントが発生すると、指定されたサブルーチンが呼び出されて実行されるのだ。

割り込み処理には以下の5つの種類がある。

・ファンクションキー入力割り込み
ON KEY GOSUB
指定したファンクションキーが押されたときに発生する割り込み。GOSUB以降にはF1～F10のキーそれぞれについて、任意に割り込み先を指定できる

■リスト1　トリガー入力割り込みの例

```
10 COLOR 15,4,7:SCREEN 1,0:WIDTH 32:KEYO
FF:DEFINT A-Z:R=RND(-TIME)
20 SPRITE$(0)=STRING$(8,255)
30 ON STRIG GOSUB 200:STRIG(0)ON
40 X=128:Y=106:XX=0:YY=0
100 '  main  --------------------------
110 X=X+XX:XX=XX+INT(RND(1)*3-1)
120 Y=Y+YY:YY=YY+INT(RND(1)*3-1)
130 IF X<0 THEN X=0:XX=-XX ELSE IF X>248
 THEN X=248:XX=-XX
140 IF Y<0 THEN Y=0:YY=-YY ELSE IF Y>184
 THEN Y=184:YY=-YY
150 IF ABS(XX)>8 THEN XX=XX-SGN(XX)
160 IF ABS(YY)>8 THEN YY=YY-SGN(YY)
170 PUTSPRITE 0,(X,Y),8,0
180 GOTO 110
200 '  wait sub  ---------------------
210 STRIG(0)OFF:BEEP
220 LOCATE 11,11:PRINT "** WAIT **"
230 IF STRIG(0) THEN 230 ELSE CLS
240 STRIG(0)ON:RETURN
```

・CTRL＋STOPキー入力割り込み
ON STOP GOSUB
CTRLキーとSTOPキーが押されたときに発生する割り込み。通常はプログラムを中断するが、この割り込みが定義されていると、プログラムを中断せずに、指定したサブルーチンが実行される

・スプライト衝突割り込み
ON SPRITE GOSUB
画面に表示されているスプライトのいずれかが重なったときに発生する割り込み。スプライトを使用したゲームなどで、よく衝突判定に使われている

・トリガー入力割り込み
ON STRIG GOSUB
STRIG関数で認識できるキーまたはボタンが押されたときに発生する割り込み。キーまたはボタンの種類によって、それぞれに任意の割り込み先を指定できる

・インターバルタイマ割り込み
ON INTERVAL GOSUB
一定の間隔で発生する割り込み。割り込みの間隔は、

INTERVAL＝30

のようにして、60分の1秒単位で指定する。この場合、60分の30、つまり2分の1秒ごとに割り込みが発生することになる

ON～GOSUB
～の部分に指定された数式の値により、GOSUB以降に指定された行番号のサブルーチンを呼び出す。数式が負の値だとエラーとなり、0または指定した行番号の個数を超える値の場合は、サブルーチンを呼び出さずに、直後の命令に移行する。

ON KEY GOSUB～
ファンクションキー入力割り込みでは、どのファンクションキーの入力があったかで、それぞれ個別の処理を呼び出すように指定できる。これは～で指定する行の順番で決まっていて、例えば、
ON KEY GOSUB 10,20,,30
となっている場合、F1キーでは行10が、F2キーでは行20が呼び出される。また、F3キーに対応する行の指定がないが、使わないキーの指定は省略することができる。

ON STRIG GOSUB～
トリガー入力割り込みでも、ファンクションキー入力割り込みとおなじように対応するイベントが複数存在する。それぞれがどの行を呼び出すかの指定方法もおなじで、呼び出す行番号の順番で決まっている。この順番は、
1番目：スペースキー
2番目：ポート1のトリガーA
3番目：ポート2のトリガーA
4番目：ポート1のトリガーB
5番目：ポート2のトリガーB
となっている。こちらも使わないキーまたはボタンに対する行の指定は省略することができる。

■リスト2　ファンクションキー入力割り込みの例

```
10 COLOR 15,4,7:SCREEN 1,0:WIDTH 32:KEYO
FF:DEFINT A-Z:R=RND(-TIME)
20 SPRITE$(0)=STRING$(8,255)
30 ON KEY GOSUB 200:KEY(1)ON
40 X=128:Y=106:XX=0:YY=0
100 '  main ------------------------
110 X=X+XX:XX=XX+INT(RND(1)*3-1)
120 Y=Y+YY:YY=YY+INT(RND(1)*3-1)
130 IF X<0 THEN X=0:XX=-XX ELSE IF X>248
 THEN X=248:XX=-XX
140 IF Y<0 THEN Y=0:YY=-YY ELSE IF Y>184
 THEN Y=184:YY=-YY
150 IF ABS(XX)>8 THEN XX=XX-SGN(XX)
160 IF ABS(YY)>8 THEN YY=YY-SGN(YY)
170 PUTSPRITE 0,(X,Y),8,0
180 GOTO 110
200 '  wait sub  ---------------------
210 KEY(1)OFF:BEEP
220 LOCATE 11,11:PRINT "HIT SPACE!"
230 IF STRIG(0)=0 THEN 230 ELSE CLS
240 KEY(1)ON:RETURN
```

割り込み処理の具体例

リスト1はトリガー入力割り込みのサンプルで、付録ディスクに
SAMPLE.SB2
というファイル名で収録しているので、実行してみてほしい。

リスト1を実行すると、画面を赤い四角がうろつき出す。そこで、適当なところでスペースキーを押してみると、トリガー入力割り込みが発生し、画面に
＊＊　WAIT　＊＊
と表示して、四角の動きが止まる。スペースキーを放すとメッセージが消えて、また四角が動き出す。

リスト1は大きく分けて3つの部分からできている。

まず、行10〜40までの初期設定の部分。ここはRUN直後に一度だけ実行される。トリガー入力割り込みもここで設定されていて、行30にある、
ON STRIG GOSUB 200
で、スペースキーの入力があったときに、行200を呼び出すように指定しているのだ。

次に行100〜180のメイン部分。赤い四角のスプライトを動かしているだけの、単純なものだ。

最後が行200〜240の割り込み処理の部分。ここは割り込みが発生するまで実行されることがない。

リスト1では、初期設定で画面や割り込みを設定したあとは、ずっとメイン部分を繰り返している。ここでスペースキーが押されると、メイン部分のどこが実行されていようと、すぐさま割り込み処理を呼び出して実行する。割り込み処理が終了すれば、またメイン部分にもどるのだ。

ON／OFF／STOP

トリガー入力割り込みは行30で設定されているが、これだけでは割り込みは発生しない。その直後にある、
STRIG(0)ON
が実行されて、初めて割り込みが有効になるのだ。

似たようなものが、割り込み処理の始めにもある。行210の、
STRIG(0)OFF
というのがそれで、これは割り込みを禁止する命令で、実行されると割り込みが発生しなくなるのだ。

もうひとつ、
STRIG(0)STOP
というのがある。これは、割り込みの発生を保留する命令で、この間に発生した割り込みの回数だけを記憶しておき、割り込みが許可されると、そこでようやく割り込み処理が呼び出されるのだ。

■リスト3　インターバルタイマ割り込みの例

```
10 COLOR 15,4,7:SCREEN 1,0:WIDTH 32:KEYO
FF:DEFINT A-Z:R=RND(-TIME)
20 SPRITE$(0)=STRING$(8,255)
30 ON INTERVAL=60 GOSUB 200:INTERVALON
40 X=128:Y=106:XX=0:YY=0:TM=0
100 '  main ------------------------
110 X=X+XX:XX=XX+INT(RND(1)*3-1)
120 Y=Y+YY:YY=YY+INT(RND(1)*3-1)
130 IF X<0 THEN X=0:XX=-XX ELSE IF X>248
 THEN X=248:XX=-XX
140 IF Y<0 THEN Y=0:YY=-YY ELSE IF Y>184
 THEN Y=184:YY=-YY
150 IF ABS(XX)>8 THEN XX=XX-SGN(XX)
160 IF ABS(YY)>8 THEN YY=YY-SGN(YY)
170 PUTSPRITE 0,(X,Y),8,0
180 GOTO 110
200 '  wait sub  ---------------------
210 INTERVALOFF:BEEP:TM=TM+1
220 LOCATE 11,11:PRINT TM;"秒 ｹｲｶ"
230 INTERVALON:RETURN
```

■リスト4　スプライト衝突割り込みの例

```
10 COLOR 15,4,7:SCREEN 1,0:WIDTH 32:KEYO
FF:DEFINT A-Z:R=RND(-TIME)
20 SPRITE$(0)=STRING$(8,255)
30 ON SPRITE GOSUB 200:SPRITE ON
32 PUTSPRITE 1,( 30, 30),15,0
34 PUTSPRITE 2,( 30,150),15,0
36 PUTSPRITE 3,(200, 30),15,0
38 PUTSPRITE 4,(200,150),15,0
40 X=128:Y=106:XX=0:YY=0
100 '  main ------------------------
110 X=X+XX:XX=XX+INT(RND(1)*3-1)
120 Y=Y+YY:YY=YY+INT(RND(1)*3-1)
130 IF X<0 THEN X=0:XX=-XX ELSE IF X>248
 THEN X=248:XX=-XX
140 IF Y<0 THEN Y=0:YY=-YY ELSE IF Y>184
 THEN Y=184:YY=-YY
150 IF ABS(XX)>8 THEN XX=XX-SGN(XX)
160 IF ABS(YY)>8 THEN YY=YY-SGN(YY)
170 PUTSPRITE 0,(X,Y),8,0
180 GOTO 110
200 '  wait sub  ---------------------
210 SPRITE OFF:BEEP
220 X=X-XX:Y=Y-YY:XX=-XX:YY=-YY
230 PUTSPRITE 0,(X,Y),8,0
240 SPRITE ON:RETURN
```

その他の割り込み処理

リスト2〜4は、リスト1を改造して、他の割り込み処理に変えてみたものだ。どれもメイン部分はまったくおなじで、行30の割り込みの設定と、行200以降の割り込み処理だけが変更されている。

リスト2はファンクションキー入力割り込みの例で、F1キーを押すと割り込みが発生し、スペースキーを押すことで、割り込みを終了するようになっている。

リスト3はインターバルタイマ割り込みの例で、画面中央付近に、プログラムを実行してからどれくらいの時間がたったかを表示する。

リスト4はスプライト衝突割り込みの例で、白い四角と赤い四角がぶつかると、メッセージを表示して、赤い四角が跳ね返るようになっている。

リスト2〜4は付録ディスクに収録できなかったが、リスト1をもとに改造して試してみて欲しい。

【リスト1解説】
●初期設定部分
10　画面の色設定／画面モードとスプライトのサイズを設定／1行に表示できる文字数を32に設定／ファンクションキーの内容表示を禁止／変数を整数型に宣言／乱数系列の初期化
20　四角のスプライトパターン定義
30　トリガー入力割り込み設定※スペースキーが押されたときは行200を呼び出すように指定する／トリガー入力割り込みの許可
40　赤い四角の座標と移動増分設定
●メイン部分
100　ＲＥＭ文※コメント行
110〜120　赤い四角の移動計算
130〜140　画面の端での跳ね返り
150〜160　移動増分の調整
170　赤い四角の表示
180　行110へ飛ぶ
●割り込み処理部分
200　ＲＥＭ文
210　トリガー入力割り込みを禁止する※割り込み処理をしているときは、割り込みが起こらないようにするため／ビープ音
220　メッセージ表示
230　スペースキーの状態判定⇨押されたままなら行230、つまりこの判定を繰り返す。放されていれば画面を消す※メッセージの消去
240　トリガー入力割り込み許可※この割り込み処理の始めで禁止にしているので許可する／もとの処理にもどる

SUPER BEGINNERS' 講座

超初心者

スーパー ビギナーズ

第10回 今回のテーマ 割り込みサブ(2)

割り込み処理でサブルーチンを呼び出す。これがいったい何の役に立つのだろう？　今回は割り込みの利用方法と、その際の注意について解説する。

前回は、割り込み処理を使うと、特定の状況やタイミングでサブルーチン処理を呼び出せる、ということを紹介した。

割り込みは全部で5種類あり、それぞれで条件も違っていた。

しかしこれらの割り込み処理は、いったいどのようなことに利用できるのだろうか？

割り込み処理の使い道

例えばシューティングゲームを作る場合で考えてみよう。

自機のミサイルや弾の発射にはトリガー入力割り込みが使える。敵への命中判定や自機のダメージ判定には、スプライト衝突割り込みを利用することができる。

インターバルタイマ割り込みは、ゲーム中のBGM演奏や敵の移動、背景のスクロール、プレイ時間の計測など、幅広く利用できる。

ファンクションキー割り込みはゲームのポーズ(一時停止)とか、オプションコマンドなどに利用することができるだろう。

少し極端な例だったかもしれないが、このように割り込み処理を使うことで、結果的に処理ごとの役割を明確にでき、メイン部分の負担を軽くすることができる。

発射処理のサンプル

右のリスト1は、トリガー入力割り込みを使ったミサイル発射処理のサンプルだ。付録ディスクに収録されていないが、かんたんなプログラムなので、ぜひ入力してみて欲しい。

実行すると、画面に白い四角が現れる。これが自機で、カーソルキーの左右で移動する。スペースキーで赤いミサイルが発射され、スペースキーを放すまで一時停止するようになっている。

まあ、とても単純なプログラムではあるが、このリストをもとに、割り込みのタイミングと関連する問題について解説しよう。

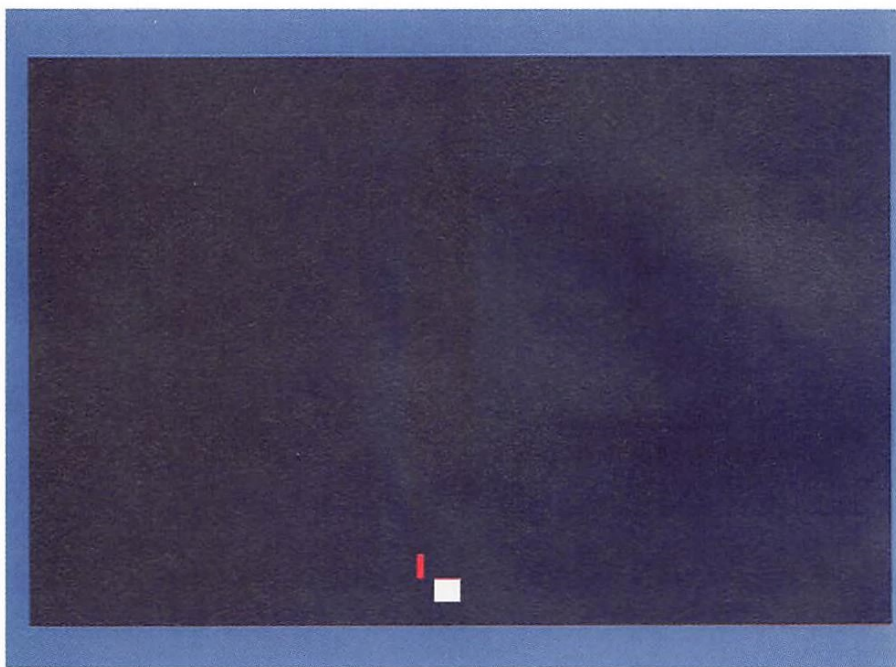
割り込みのタイミングと問題

リスト1を実行して、いろいろ試してみると、右の画面のような状況になることがある。

ミサイルが自機の真上ではなく、ずれた位置に表示されるのだ。

これは何もわざとそうしているわけではなく、割り込みが起こるタイミングで起こってしまう問題なのだ。この原因の説明をしよう。

■リスト1　トリガー割り込みを使った発射処理

```
10 COLOR 15,4,7:SCREEN 1,0:WIDTH 32:KEYO
FF:DEFINT A-Z:R=RND(-TIME)
20 SPRITE$(0)=STRING$(8,255)
30 SPRITE$(1)=STRING$(8,24)
40 ON STRIG GOSUB 200:STRIG(0)ON
50 X=128:YY=-9
100 '  main  --------------------------
110 S=STICK(0)
120 X=X+(S=7)*8-(S=3)*8
130 IF X<0 THEN X=0 ELSE IF X>248 THEN X
=248
140 PUTSPRITE 0,(X,175),15,0
150 IF YY=>-1 THEN YY=YY-8
160 PUTSPRITE 1,(XX,YY),8,1
170 GOTO 110
200 '  fire sub  ----------------------
210 STRIG(0)OFF:BEEP:XX=X:YY=167
220 PUTSPRITE 1,(XX,YY),8,1
230 IF STRIG(0) THEN 230
240 STRIG(0)ON:RETURN
```

↑移動しているときにミサイルを発射すると、ずれた位置にミサイルが表示されることがある

特定の状況やタイミングで
割り込みの条件には、ファンクションキーが押されたときに起こるファンクションキー入力割り込みを始めとして、スペースキーや、ジョイスティックやマウスのボタン入力を使ったトリガー入力割り込み、スプライトが重なったときに起きるスプライト衝突割り込み、内蔵タイマーを使って一定間隔ごとに呼び出すインターバルタイマ割り込み、ＣＴＲＬ＋ＳＴＯＰキーが押されたときに起きるＣＴＲＬ＋ＳＴＯＰキー入力割り込みがある。

メイン部分の負担を軽くする
ミサイルの発射判定や命中判定などを割り込み処理で行えば、メイン部分で行うことが少なくなる。そうなると、１回の処理に必要な時間も当然少なくなるので、全体のスピードアップにもつながるのだ。

連射を防ぐ
実際にはミサイルが複数現れるわけではないので、正しい表現ではないが、すでに発射しているミサイルが画面にあるときに割り込みが起きると、そのミサイルは消えて、新たにミサイルが発射されるのを防ぐのだ。

ＣＴＲＬ＋ＳＴＯＰキー入力割り込みを使う場合の注意
ＣＴＲＬ＋ＳＴＯＰキー入力割り込みを使ったプログラムを作成する場合、実行する前に必ず保存しておくのを忘れないようにしよう。例えば、

```
10 ON STOP GOSUB 30:STOP ON
20 GOTO 20
30 RETURN
```

のように、割り込み処理でプログラムを終了することなく、継続するようになっている場合、プログラムを終了することができなくなり、リセットして悲しい思いをすることがあるからだ。

■リスト2　STRIG(Ø)STOPを使って改良

```
10 COLOR 15,4,7:SCREEN 1,0:WIDTH 32:KEYO
FF:DEFINT A-Z:R=RND(-TIME)
20 SPRITE$(0)=STRING$(8,255)
30 SPRITE$(1)=STRING$(8,24)
40 ON STRIG GOSUB 200
50 X=128:YY=-9
100 '  main -------------------------
110 S=STICK(0)
120 X=X+(S=7)*8-(S=3)*8
130 IF X<0 THEN X=0 ELSE IF X>248 THEN X
=248
140 PUTSPRITE 0,(X,175),15,0
145 STRIG(0)ON
150 IF YY=>-1 THEN YY=YY-8
160 PUTSPRITE 1,(XX,YY),8,1
170 STRIG(0)STOP:GOTO 110
200 '  fire sub ---------------------
210 STRIG(0)OFF:BEEP:XX=X:YY=167
220 PUTSPRITE 1,(XX,YY),8,1
230 IF STRIG(0) THEN 230
240 RETURN
```

■リスト3　連射を防ぐための改良

```
10 COLOR 15,4,7:SCREEN 1,0:WIDTH 32:KEYO
FF:DEFINT A-Z:R=RND(-TIME)
20 SPRITE$(0)=STRING$(8,255)
30 SPRITE$(1)=STRING$(8,24)
40 ON STRIG GOSUB 200
50 X=128:YY=-9
100 '  main -------------------------
110 S=STICK(0)
120 X=X+(S=7)*8-(S=3)*8
130 IF X<0 THEN X=0 ELSE IF X>248 THEN X
=248
140 PUTSPRITE 0,(X,175),15,0
150 IF YY=>-1 THEN YY=YY-8 ELSE STRIG(0)
ON
160 PUTSPRITE 1,(XX,YY),8,1
170 STRIG(0)STOP:GOTO 110
200 '  fire sub ---------------------
210 STRIG(0)OFF:BEEP:XX=X:YY=167
220 PUTSPRITE 1,(XX,YY),8,1
230 IF STRIG(0) THEN 230
240 RETURN
```

位置がずれる原因

プログラムをRUNすると、まず行50までの初期設定が実行される。そこでトリガー入力割り込みの設定と許可も行われている。

行100～170のメイン処理では、スティック入力による自機の操作と移動・表示、そしてミサイルの移動と表示だけが行われる。

つまりメイン部分では、始めにトリガー入力割り込みが許可されたまま、ずっと有効になっているというわけだ。そして、じつはこのことが問題の原因なのだ。

スティック入力を受け付けて、行120で自機の座標を計算した直後にトリガー入力があった場合、自機の表示を更新する前に割り込みが起きることになる。

すると、自機の座標と表示が合っていない状態で、ミサイル発射処理のサブルーチンが呼び出されてしまうのだ。ミサイル発射処理では、自機の座標からミサイルの座標を決定し、ミサイルを表示してもどる。この瞬間、自機の表示位置とミサイルの表示位置がずれてしまうのだ。

実際にはメイン処理にもどったときに、自機の表示も更新されるので、このような問題があってもあまり気付かないことが多い。

しかし、割り込み処理を使ったプログラムではよく起こる問題だ。

タイミングを限定して改良する

では、このような問題があった場合、いったいどうすればいいのだろうか？

答えは単純で、割り込みが起こると困る部分では、割り込みをさせなければいいのだ。言い換えると「割り込み許可の部分を限定する」のである。

そこで、リスト1をそのように改良したのが、左上のリスト2だ。

リスト2での変更点は、行40の終わりと行240の始めにあった、

STRIG(0)ON

を削除して、行145に移動させていることと、行170の始めに、

STRIG(0)STOP

を置いていることだ。

これで、自機の移動計算と表示の間は割り込みが起こらないので、問題は解決される。

また、STOPで保留しているあいだにスペースキーが押された場合、まったく無視するわけではなく、許可になりしだい、割り込みを実行するので、入力漏れになるおそれもない。

■リスト4　CTRL＋STOP割り込みの例(その1)

```
10 ON STOP GOSUB 100
20 STOP ON:CLS
30 LOCATE 10,10:PRINT INT(RND(1)*6)+1
40 GOTO 30
100 STOP OFF
110 PRINT "CTRL+STOPｶﾞ ｵｻﾚﾏｼﾀ."
120 PRINT:PRINT"ｽﾍﾟｰｽｷｰﾃﾞ ｼｭｳﾘｮｳｼﾏｽ."
130 IF STRIG(0)=0 THEN 130
140 END
```

■リスト5　CTRL＋STOP割り込みの例(その2)

```
10 ON STOP GOSUB 100
20 STOP ON:CLS
30 LOCATE 10,10:PRINT INT(RND(1)*6)+1
40 GOTO 30
100 PRINT "CTRL+STOPｶﾞ ｵｻﾚﾏｼﾀ."
110 PRINT:PRINT"ｽﾍﾟｰｽｷｰﾃﾞ ｼｭｳﾘｮｳｼﾏｽ."
120 IF STRIG(0)=0 THEN 120
130 STOP OFF
140 END
```

ON／OFF／STOPの位置

右上のリスト3は、リスト2の行145と行150をひとつにまとめ、ミサイルが表示されているときは、割り込みが起こらないように改良したものだ。これにより、連射を防ぐことができる。

リスト4と5は、前回はサンプルを省略したCTRL＋STOPキー入力割り込みの例だ。

2つの違いは、割り込み禁止が置かれている場所だ。プログラムを実行して、CTRL＋STOPキーを押してみよう。割り込みが起きると、メッセージを表示して、スペースキーの入力待ちになるのだが、ここでもう一度CTRL＋STOPキーを入力すると、それぞれの違いがよくわかるはずだ。

このように、割り込みの許可や禁止命令の位置をちょっと変えるだけで、ずいぶん違いがあるのがわかるだろう。これは割り込みを受け付ける部分に限らず、呼び出されるサブルーチン処理においても例外ではない。

割り込み処理の難しいところは、この許可、禁止、保留命令を置く場所にある。なにか問題があった場合、そのほとんどが、これらの配置に関係していることが多いので、よく覚えておいてほしい。

【リスト1の解説】

●初期設定部分

10　画面の色設定／画面モードとスプライトのサイズを設定／1行に表示できる文字数を32文字に設定／ファンクションキーの内容表示を禁止／変数をすべて整数型に宣言／乱数初期化
20　自機(白い四角)のスプライトパターン定義
30　ミサイル(赤い縦線)のスプライトパターン定義
40　トリガー入力割り込み設定※スペースキーが押されたら行200を呼び出す／トリガー入力割り込みの許可
50　自機のX座標設定／ミサイルのY座標設定

●メイン部分

100　REM文※コメント行
110　スティック入力受け付け
120　自機のX座標の移動計算
130　自機のX座標が画面の外に出たら座標を調整する
140　自機のスプライトを表示
150　ミサイルが画面内に表示されているかの判定⇨表示されているなら、ミサイルのY座標の移動計算
160　ミサイルのスプライトを表示
170　行110へ飛ぶ※メイン部分を繰り返す

●ミサイル発射サブ(割り込みサブ)

200　REM文
210　トリガー入力割り込みを禁止する※割り込み処理をしているときに割り込みが起こらないようにするため／効果音(ビープ)／ミサイルの初期座標を設定※自機の真上に配置
220　ミサイルのスプライトを表示
230　スペースキーが押されたままならこの行を繰り返す
240　トリガー入力割り込みを許可する※割り込み処理の始めで禁止しているので許可しておく／もとの処理にもどる

SUPER BEGINNERS'講座

超初心者

スーパー ビギナーズ

第11回
今回のテーマ
ページ切り換え

ファンダムの採用作品『1スクリーンファイト2』では、ページ切り換えのテクニックを使ってキャラの動きを表現している。このテクニックについて紹介しよう。

ページ切り換え

ページ切り換えというと、多くの領域を一定の大きさに区切ったものをページと呼び、どれを使うか切り換える作業をいう。

今回のSBで紹介するページ切り換えとは、MSXの画面表示に関する切り換えだ。

画面表示に関する切り換えでは、アクティブページの切り換えと、ディスプレイページの切り換えの2つがあるので、覚えておこう。

ディスプレイページ

大きな紙を折り畳み、その1面ごとに番号を振ったものを想像してほしい。

今、番号1の面が見えているとする。次に、折り返して2の面を表に向けたとしよう。

見えている面、つまりページが1から2に変わる。これがディスプレイページの切り換えだ。

ディスプレイページとは、画面に表示されているページのことなのだ。

アクティブページ

アクティブページというのは、ちょっとややこしい。

さきほどの折り畳んだ紙を例にすると、見えている面はそのままにして、違う面に文字を書いたりするとしよう。

実際に文字を書いたりする面が変わる。これがアクティブページの切り換えだ。

アクティブページとは、実際に絵や文字を書くページのことをいうのだ。

このように、画面に表示されるページと、絵などを描く画面とが分かれているために、初心者にはとっつきにくいものがある。

しかし、2つのページをうまく使えば、画面にはでき上がった絵などを表示しておいて、その裏で、違うページに次の絵を描いておく、などということができるのだ。

こうすることで、描いている途中を見せることなく、スムーズに表示を切り換えていけるわけだ。

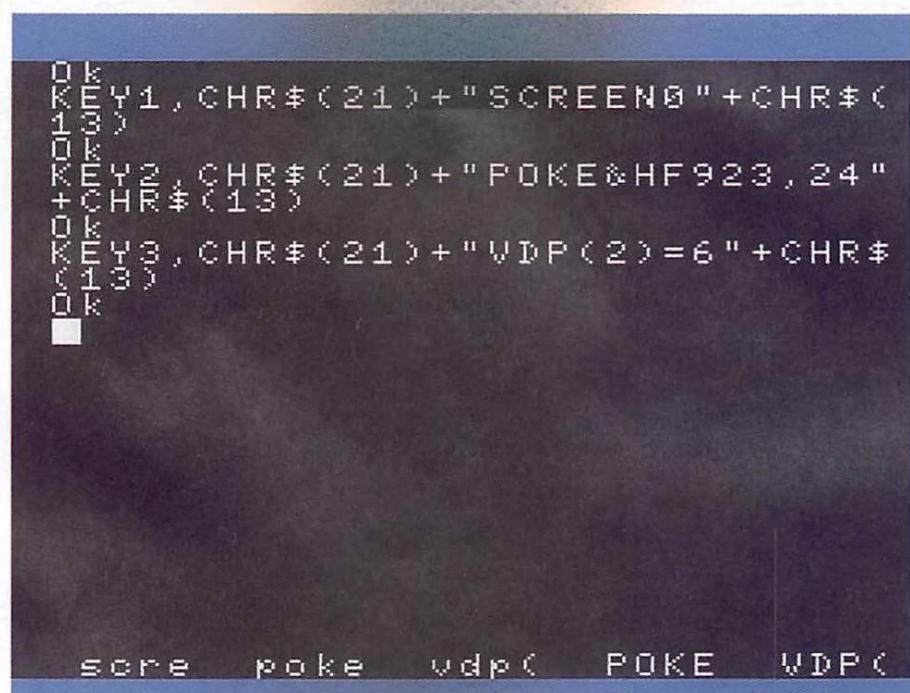
画面モードによる違い

SCREEN5以降の画面モードでは、SETPAGEという命令を使って、2つのページをかんたんに切り換えることができる。

しかし、SCREEN4までの画面モードには、ページ切り換えのための命令がない。

そこで、SCREEN4までの画面モードでは、ちょっと特殊なやり方をして、ページ切り換えを行う必要があるのだ。

```
Ok
KEY1,CHR$(21)+"SCREEN0"+CHR$(
13)
Ok
KEY2,CHR$(21)+"POKE&HF923,24"
+CHR$(13)
Ok
KEY3,CHR$(21)+"VDP(2)=6"+CHR$
(13)
Ok
█

 scre   poke   vdp(   POKE   VDP(
```

↑SCREEN1に変更した画面。アクティブページとディスプレイページが同じなので、キーボードから入力した文字が、すぐ画面に表示されている

SCREEN1の画面の場合

では実際に、SCREEN1の画面でページ切り換えをする方法を紹介しよう。

SCREEN1の画面は、上の写真のように、文字が表示されるテキスト画面だ。

画面モードを変えたばかりなので、ディスプレイページとアクティブページは同じで、キーボードから入力した文字は、そのまま、見えている画面に表示される。

ではもし、アクティブページとディスプレイページを違うページに指定するとどうなるだろうか？

キーボードから入力した文字が、今見えている画面には表示されないことになる。これでは試したら最後、元にもどすのに、たいへん苦労してしまう。

その他のページ切り換え
裏RAMという言葉を見たことがあると思う。メインRAMが64K以上搭載されているMSXでは、BASICで使えるRAMは32Kに固定されている。そこでRAMのページ切り換えということをして、使えるRAMを増やすことがある。このように、画面表示以外のところでも、ページ切り換えは存在するのだ。

2つのページ
画面モードを変更した直後では、2つのページは同じに初期化される。そのため、PRINT文で文字を書いたり、LINEなどで絵を描くと、見えている画面に表示されるのだ。

SETPAGE
SCREEN5以降の画面モードで有効な、ページを切り換えるための命令がSETPAGEだ。この書式は、

SETPAGE　n，m
となっていて、nにディスプレイページの番号を、mにアクティブページの番号を指定する。片方の指定を省略することもできるが、その場合、省略されたページは切り換わらない。

苦労してしまう
入力した文字が画面に表示されないので、ミスがあってもわからない。また、アクティブページの指定によっては、VRAMに保存されている情報が文字となって画面に出ている場合がある。このようなページでコマンドを入力しても、前後に文字があるために、ちゃんと入力できても、うまくいかないこともあるのだ。

ファンクションキーの設定
ファンクションキーに設定しておけば、入力した文字が見えないための苦労をすることもなく、試すことができる。

■ファンクションキーの設定

```
KEY1, CHR$(21)+"SCREEN0"+CHR$(13)
KEY2, CHR$(21)+"POKE&HF923,24"+CHR$(13)
KEY3, CHR$(21)+"VDP(2)=6"+CHR$(13)
```

■アクティブページの切り換え

```
POKE&HF923,〈ページ番号〉×4
```

■ディスプレイページの切り換え

```
VDP(2)=〈ページ番号〉
```

そこで、あらかじめ、ファンクションキーに、元にもどす設定をしておいてほしい。

設定するファンクションキーは、F1～F3の場合、上のようにしておけばOKだ。この設定により、以下のような働きをする。

・F1キー　画面モードをSCREEN0にする

・F2キー　アクティブページを元のページにもどす

・F3キー　ディスプレイページを元のページにもどす

ページを切り換え過ぎて、めちゃくちゃになってしまったら、これらのファンクションキーで復活させればよい。

なお、いろいろ試していると、妙なスプライトが表示されたり、一部の文字の形が壊れたりするときがある。

そういった場合には、F1キーで画面モードをSCREEN0にして、それからSCREEN1に戻せば、スプライトも文字の形も初期状態にもどるので、安心して試してほしい。

アクティブページの切り換え

SCREEN1でアクティブページを切り換えるには、POKE文を使って、ワークエリアを直接書き換える必要がある。

やり方は上にあるとおりで、ページ番号×4の値を書き込むことで行う。

なお、SCREEN1で指定できるページ番号は、0～15の範囲までだ。

また、POKE文で指定するアドレス(&HF923)は、間違わないように注意。

ディスプレイページの切り換え

ディスプレイページを切り換えるには、上にあるように、VDP関数を使って行う。

こちらのほうも、VDP関数のカッコの中には、必ず2を指定するように注意してほしい。

SCREEN1のVRAM

切り換えてみるとわかるが、あらかじめ文字が書き込まれているページがいくつかある。

これらは、VRAMに保存されている文字のパターンデータや、色、スプライトの情報などが、文字として画面に表示されるのだ。

VRAMがどのようになっているかを知っていないと、このような情報がある部分を壊してしまい、文字の形がおかしくなったり、妙なスプライトが出たりするのだ。

次回はVRAMの説明を含めて、より詳しく紹介しよう。(MORO)

```
Ok
KEY1,CHR$(21)+"SCREEN0"+CHR$(
13)
Ok
KEY2,CHR$(21)+"POKE&HF923,24"
+CHR$(13)
Ok
KEY3,CHR$(21)+"VDP(2)=6"+CHR$
(13)
Ok
KEY4,CHR$(21)+"POKE&HF923,4*"

Ok
KEY5,CHR$(21)+"VDP(2)="
Ok

 scre   poke   vdp(   POKE   VDP(
```

↑アクティブページを変えてみた。キーボードから入力した文字は、画面に表示されない

↑ディスプレイページを0にしてみた。データが文字になって、画面にたくさん表示された

できれば、F4、F5キーに、以下のような設定をしておけば、もっと楽にページ切り換えを行えるだろう。

```
KEY4, CHR$(21)+"POKE&HF923, 4*"
KEY5, CHR$(21)+"VDP(2)="
```

キーを押した後、ページ番号を入力してリターンキーを押すと、F4キーでアクティブページが、F5キーでディスプレイページが切り換わる。

情報がある部分

アクティブページを0や1にして、キーボードから文字を入力してみよう。文字のパターン情報が壊されて、文字の形が変わっていくはずだ。このように、VRAMに保存されている情報を、PRINT文を使って文字を表示することで、直接書き換えるというテクニックもあるので、覚えておくといいかも知れない。

プログラムご相談・修理・仕立て MORO'Sショップ

●文字の色つけについて

BASICでアドベンチャーゲームを作っているのですが、SCREEN1で、「どうする?」「なにを?」などのメッセージだけを、他のメッセージと違う色にする方法はありませんか？（愛知県・DON／／17歳）

SCREEN1の場合、8文字ごとに色をつけることができますが、特定の行だけの色つけはできません。でも、色をつけなくても、メリハリはつけられると思いますよ。例えば、メッセージの先頭に、「>」や「●」をつければ、だいぶ違うはずですし、その部分だけ、点滅させるという方法もあります。色つけにこだわらず、工夫されることをすすめます。

●RUN命令について

SOUND文とPLAY文を組み合わせて使っていたときのことです。プログラムをRUNでリプレイしたとき、音が変になってしまいました。ダイレクトにRUNを使う場合と、プログラム中でRUNを使う場合とでは、どのような違いがあるのでしょうか？（大阪府・まこりん／?歳）

※質問の内容が長文でしたので、要約して記載しました

両者の機能に違いはありません。手紙には詳しい状況が書かれていたので、そこから推測できる原因は、CTRL＋STOPでプログラムを中断したときに鳴るビープ音の影響と思われます。ビープ音もPSGで鳴るひとつの効果音です。そのため、ビープ音を鳴らした後では、PSGのレジスタが変化してしまうのです。試しに、問題のプログラムの先頭かリプレイの部分に、BEEPを入れて実行してみてください。

☆お知らせです

神奈川県の鈴木さんからのご依頼「バイオリズムのプログラム」を始め、プログラム作成が遅れていましたが、次号の付録ディスクに収録する予定です。

ご依頼の数が思ったよりも多く、まだすべてを処理できていませんが、作業は行っていますので、もうしばらくお待ちくださいませ。（店長）

SUPER BEGINNERS' 講座

スーパー ビギナーズ

超初心者

第12回

今回のテーマ
VRAMとページ切り換え

ページ切り換えを駆使するには、どうしてもVRAMの知識が必要になってくる。今回はページ切り換えに関係するSCREEN1のVRAMについて紹介する。

ページ切り換えを使うためには、どうしてもVRAMに関連した知識が必要になってくる。しかし、たいていの初心者は、VRAMという言葉を見ただけで後込みしてしまう。でも、安心して欲しい。ページ切り換えで必要になるVRAMの知識なんて、それほど多くはないのだから。

パターン名称テーブル

ページ切り換えで大切なのは、なんといってもパターン名称テーブル。これさえ理解していれば、ページ切り換えを使うことができるからだ。

パターン名称テーブルとは、画面のどこに、どんな文字があるかの情報が記録されている領域だ。SCREEN1の画面では、最大で32文字×24行の表示が可能なのは知っていると思う。この画面に表示されている1つ1つの文字が、順番に文字コードの形で記録されているのだ。空白も、WIDTH文の設定によって使わなくなった端の部分も、ファンクションキーの内容が表示されている部分も、画面に見えている文字の情報はすべて記録されている。

ということは、パターン名称テーブルには、つねに32×24文字ぶん、計768バイトの大きさの領域が必要だということになる。ページ切り換えをして、画面を他のページに切り換えるのだとしたら、そこには最低でもそれだけの領域が空いていないといけないわけだ。

アクティブページの切り換え

アクティブページとは、文字が書き込まれるページのこと。つまりこれはパターン名称テーブルを意味している。

このパターン名称テーブルが置かれている場所は、ワークエリアの&HF922番地と&HF923番地に保存されている。この2つのワークエリアに書き込まれている値を変更すると、パターン名称テーブルの場所を変えることができるのだ。

右の写真は、&HF922番地に99を書き込んで、パターン名称テーブルの位置を99バイトずらしたものだ。カーソルキーでカーソルを動かしてみたり、文字を書いてみたり、CLSしてみるとわかると思うが、画面が変なつながり方をしている。

「POKE&HF922, 99」を実行してみたところ。画面のつながり方が変になっている

では、今度は&HF922番地に64を書き込んでみよう。画面が下に2行ぶんずれた感じになるはずだ。ということは、あまり大きく変えると、画面に表示されなくなってしまうことも起こりうる。

ディスプレイページの切り換え

ところで先程は、パターン名称テーブルには画面に表示されている文字のすべてが保存されていると説明した。しかしこの状態ではそれがずれてしまっている。

じつは、VRAM、つまりパターン名称テーブルには文字の情報が保存されているが、それを実際に画面に表示するのはVDPの仕事なのだ。VDPの、といわれてもピンとこないかもしれないが、これは鵜呑みにしてかまわない。

このVDPが画面に表示する領域をディスプレイページと呼び、VDP(2)というシステム変数を使って指定する。

ディスプレイページの指定は&H400単位でしか行えない。なぜそうなのかは特に気にせずに、これも鵜呑みにしてしまおう。

順番に文字コードの形で
パターン名称テーブルに文字が保存されている順番は、一番上の行の左から順に右のほうへ、ついで次の行の左から右のほうへ……というように保存されている。文字コードで保存されているといってもグラフィックキャラクタの場合は特殊で、「月」～「小」の文字が、それぞれ1～31のコードで保存されているので注意。

WIDTH文
WIDTHとは、画面に文字を表示するときの横幅、つまり1行に表示できる文字数を設定する命令で、SCREEN1の画面では1～32の範囲で設定できる。

使わなくなった端の部分
WIDTHの設定で、31以下を指定すると、画面の端のほうに、カーソルが行けない部分ができる。LOCATE文とPRINT文を使っても、そこには文字を表示することができない。

2つのワークエリア
&HF922番地には、パターン名称テーブルが置かれている場所の下位アドレスが保存されていて、&HF923番地には上位アドレスが保存されている。ページ単位での切り換えを行う場合には、&HF923番地だけを書き換えればよい。

SCREEN1のVRAMと使えるページ

■ページ0(1)

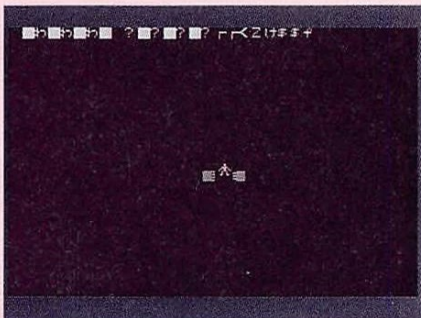
ディスプレイページを0にして、パターンジェネレータテーブルを覗いてみる

■ページ6

ディスプレイページを6にすると、いつも見慣れた画面になった

■ページ8

ページ8の最初の部分には、文字の色と、パレットの情報が記録されている

■ページ14(15)

「ザ・はさみうち!!」を遊んだあと。上部の文字は、定義されたスプライトの形の情報

使われているページ

左はSCREEN1のVRAMの使用状況を示したものだ。全16ページ中、自由に使えるページは11ページもある。

0～1と8、14～15の5ページには、文字やスプライトのパターンの情報や、パレットや文字の色の情報が保存されている。これらの場所にページ切り換えをして文字を書き込んだりすると、保存されている情報が壊されて、文字の形がおかしくなったり、色が変になったりしてしまう。

ただし、ディスプレイページをこれらのページに切り換えても、保存されている情報が壊されたりすることはないので、上の写真のように、ちょっと覗いて見るぶんにはいいだろう。

はじめはなんだかわからなくても、何度も見ているうちに、けっこう理解できたりするものだ。

ゲームで遊んだ後にでも、ちょっとやってみて欲しい。

使えるVRAM

ページ切り換えの基本は、前述したパターン名称テーブルの位置を変更して、文字を表示したりして使う画面を増やす方法だ。

パターン名称テーブルの位置は1バイト単位で変更できるのだが、ディスプレイページの変更は&H400単位でないと変更できない。そのためパターン名称テーブルを変更するときは、&H400単位で変えるのが望ましい。

SCREEN1のVRAMでは、いくつかの場所に、スプライトやパレットの情報が記録されているので、ページ切り換えをするときには注意が必要だ。まったく使われていない部分はたくさんあるので、そこを使うようにしよう。

そういったVRAMについてのことは上にまとめておいたので、そちらを参考にして欲しい。

さて次回は、ページ切り換えの具体的な方法と、ページ切り換えを使ったちょっとしたテクニックをいくつか紹介しようと思う。それまでいろいろ自分でやってみて欲しい。 (MORO)

&H400単位
10進数に直すと1024という数値になる。VDP(2)に設定する数値は、すべて&H400単位となるので、例えばディスプレイページのアドレスを&H1000にする場合、
&H1000÷&H400＝4
となるので、
VDP(2)＝4
のように指定する。

プログラムのご相談・修理・仕立て MORO'Sショップ

●キーバッファクリアってなんですか？ (石川県・MASA／18歳)

キーバッファとは、キーボードから入力された文字の情報が一時的に保存される場所のことです。キーバッファクリアとは、キーバッファに溜まった情報を消すことをいいます。たとえば、

```
10 IF STRIG(0)=0 THEN 10
```

というプログラムをRUNしてみてください。このプログラムをRUNすると、スペースキーが押されるまでじっと待っています。その間にいろいろな文字を入力して、最後にスペースキーを押してみましょう。すると画面に、今まで押した文字がいっぺんに表示されたはずです。RUNしてからスペースキーを押すまでの間に入力された文字は、すべてこのキーバッファに保存されていたのです。プログラムが終了し、カーソルが表示された時点でキーバッファから文字が送られて、画面にどっと表示されたのです。このとき、もし「NEW」などと入力していたら、プログラムが終了したとたん、リストが消えてしまうでしょう。そういったことが起きないように、キーバッファに溜まった情報を空読みして、クリアする必要が出てくるのです。

●プログラムでCAPSキーを押したことにするには、どうすればいいのでしょうか？ また、ほかのキーもあれば教えてください。 (京都府・金城健一／16歳)

CAPSキーをオンの状態にするには、

```
POKE&HFCAB, 255
```

として、ワークエリアの&HFCAB番地に255を書き込みます。逆にオフの状態にするには0を書き込みます。しかしこれだけではCAPSランプは変化しません。ランプの状態を変えるには、マシン語を組んでBIOSを呼び出す必要があります。他のキーですが、&HFCAC番地に255を書き込むと、かなキーをオンにした状態に、0を書き込むと、かなキーをオフにした状態にできます。これもまた、ランプの状態を変えるには、マシン語を組む必要があります。

☆どう組むか、どう直せばいいか、どうすればうまくいくかといった、プログラムに関するご相談・修理・仕立てのご注文を受け付けています。「SB講座MORO店」まで送ってください。質問は誌面にて、プログラムのご依頼は付録ディスクにてお答えします。

SUPER BEGINNERS' 講座

超初心者

スーパー ビギナーズ

第13回

今回のテーマ

ページ切り換えを使う

プログラムでページを切り換えるとき、どんなことに注意すればいいか？　また、ページ切り換えを使えば、どんなことが可能なのか？

前回までは、あらかじめファンクションキーに登録しておいたり、キーボードから直接入力したりしてSCREEN1のページ切り換えを体験してきた。

では、プログラムでページ切り換えを使いたいとき、どのようにしておこなえばいいだろうか？

POKE文とVDP(2)を使って、その場その場で切り換える。

もちろんそれでいいのだが、プログラムの中で、何度もページを切り換えたりするときは、いっそサブルーチンにでもしておいたほうがやりやすい。

今回はまず、プログラムでページ切り換えを使うときについて、紹介しよう。

ページ切り換えサブを作る

次ページのリスト1とリスト2は、ページ切り換えサブルーチンの例だ。

パッと見ただけで、両者の違いがわかる人は優秀。すでにページ切り換えの難関の1つをクリアしているといえる。

2つのサブルーチンの違いは、リスト1ではページ単位、リスト2ではアドレス単位になっているという部分にある。

SCREEN1のページ切り換えでは、ディスプレイページとアクティブページとで、切り換えられる大きさが違っている。ディスプレイページでは、&H400バイトのページ単位の切り換えしかできないのに対して、アクティブページでは1バイト単位で切り換えることができる。

この違いがあるために、ページ切り換えを使う目的によっては、どちらで切り換えたほうがよいか変わってくる。

例えば、いくつかのページに文字を書いておき、それを連続して切り換えるなどというときには、ページ単位で切り換えてやったほうがやりやすいはずだ。

実際には、ディスプレイページのみとか、アクティブページのみで切り換えることが多いので、慣れてきたら、その場その場で切り換えるほうがいいだろう。

注意すること

プログラムでページ切り換えをおこなうとき、注意してほしいことが2つある。

まず、プログラムを組んでいるときの注意。ためしにRUNしたときなどで、エラーが出たりして

↑画面のプログラムを実行すると……？　ページ切り換えの後、エラーで中断してしまう

プログラムが途中で止まったとき。ディスプレイページとアクティブページが別々になっていたり、アクティブページがVRAMの文字やスプライトの情報が保存されている部分になっていたりすると、打ち込んだ文字が見えなかったり、画面の表示がおかしくなっていたりして困ってしまうことがある。

泣く泣くリセットしたという人もときどきいるので、そのようなことがないように、あらかじめ、ファンクションキーのどれかに、画面モードをSCREEN0に変更する設定を登録しておこう。

もうひとつはプログラムが完成した後だ。CTRL+STOPでプログラムの実行を中断したとき、先に述べたのとおなじ結果になることがある。CTRL+STOPが押されたときに割り込みがかかるようにしておき、その割り込み処理でページを元にもどすようにしておこう。

ページ切り換えを使う

ページ切り換えを使えば、いったいどんなおもしろいことができるのだろうか？

ここではページ切り換えを使っ

POKE文とVDP(2)

SCREEN1でページ切り換えをするには、ディスプレイページをVDP(2)に、アクティブページをワークエリアの&HF922～&HF923番地にPOKE文で設定しておこなう。結果的にページ切り換えになるだけで、これらはページ切り換えのための命令ではないので注意。

困ってしまうことがある

プログラム作成時にエラーが出て中断した場合、ページ切り換えをしているときだと、そのエラーメッセージさえも出ないことがある。エラーが出たときのビープ音を聞き逃してしまうと、エラーが出てストップしたことさえもわからないことがある。こんなときは、エラー発生時の割り込み処理を使うと便利なので覚えておこう。

SCREEN0に変更する設定

画面モードを変更すると、切り換えたページが初期化されて元の状態にもどる。そこでファンクションキーに画面モードをSCREEN0にする設定をしておくと便利だ。

KEY6, CHR$ (12)＋"SCREEN0"＋CHR$ (13)

のようにすると、F6キーに画面モードを変更する設定を登録できる。

CTRL+STOP割り込み

ON STOP GOSUB～

で、CTRL+STOPが入力されたときに呼び出す処理を指定できる。この設定を有効にするには、STOP ONを実行すればよい。割り込み処理ではページ切り換えをおこなって元の状態にもどすようにしておけば、途中で中断されたときでも、正常な画面を表示することができる。

■リスト1　ページ切り換えサブの例(その1)

```
100 ' DP:ﾃﾞｨｽﾌﾟﾚｲ・ﾍﾟｰｼﾞ AP:ｱｸﾃｨﾌﾞ・ﾍﾟｰｼﾞ
110 VDP(2)=DP
120 POKE &HF922,0:POKE &HF923,AP*4
130 RETURN
```

■リスト2　ページ切り換えサブの例(その2)

```
100 ' DA:ﾃﾞｨｽﾌﾟﾚｲ・ﾍﾟｰｼﾞ ｱﾄﾞﾚｽ
110 ' AA:ｱｸﾃｨﾌﾞ・ﾍﾟｰｼﾞ ｱﾄﾞﾚｽ
120 VDP(2)=DP ¥ 1024
130 POKE &HF922,AA MOD 256
140 POKE &HF923,AA ¥ 256
150 RETURN
```

■リスト3　ページ切り換えを使ったキャラクタパターン定義

```
10 SCREEN 1:WIDTH 32:KEYOFF
20 POKE &HF923,0
30 LOCATE 0,16:PRINT "UｪUｪUｪUｪ"
40 POKE &HF923,2
50 LOCATE 8,8:PRINT "?ｓちへへて█;ｵﾎぁウウヌ█ﾜ"
60 POKE &HF923,24
70 PRINT "  ab"
80 PRINT "@@@@@@@"
90 PRINT:END
```

たいくつかのテクニックを紹介しよう。

キャラクタパターンの定義

リスト3は、ページ切り換えを使ってキャラクタパターンを定義するサンプルだ。

ファンダムの作品でよく見かけるキャラクタパターン定義とは違って、VPOKE文などがない。あるのはアクティブページを切り換えるPOKE文とPRINT文だけだ。

アクティブページがどこに切り換えられているかわかるかな？　ページ0、つまり文字の形の情報が保存されている領域に切り換えられている。ということは、ここで何か画面に書き込むと、文字の形の情報が変化することになる。

つまり、リスト3のPRINT文で、キャラクタパターンを定義しているのだ。

リスト3を実行すると、右上の写真のような画面になる。そしてそのときのページ0の様子を、

VDP(2)=0

を実行してよく見てみると、その画面の中に、リスト3の行30で表示した文字があるはずだ。

おなじようなやり方を使って、スプライト属性テーブルを書き換えてやれば、複数のスプライトをいっぺんに動かすこともできる。

VPOKEの代わりにPRINTでVRAMの情報を書き換えるというテクニックなのだ。

スクロールを使って表示

ディスプレイページのうしろの4分の1は、画面に表示することができない。だが、MSX2以降の機種ならVDP(24)を使って表示することができるのだ。

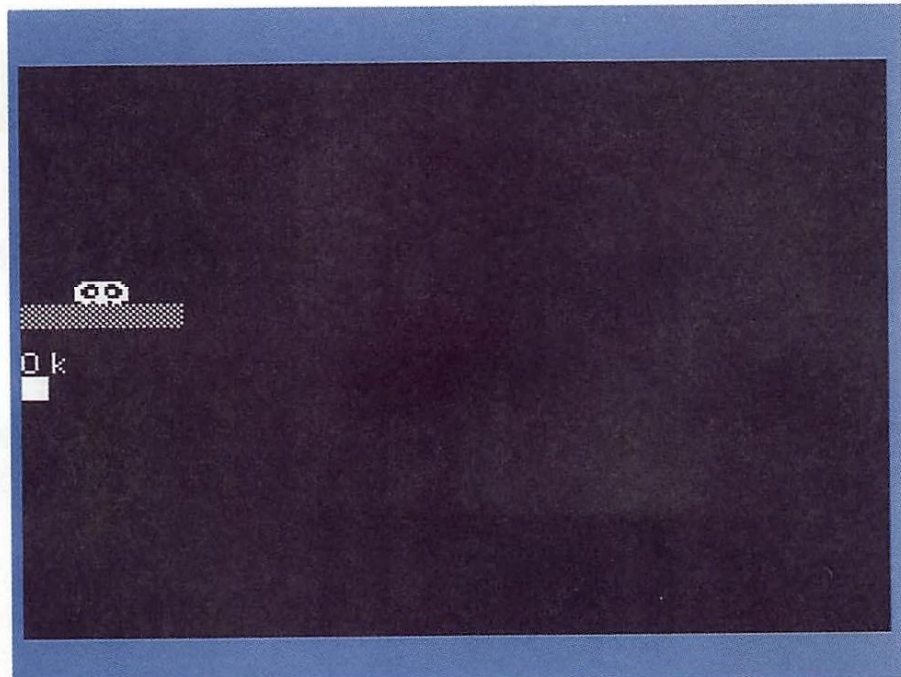
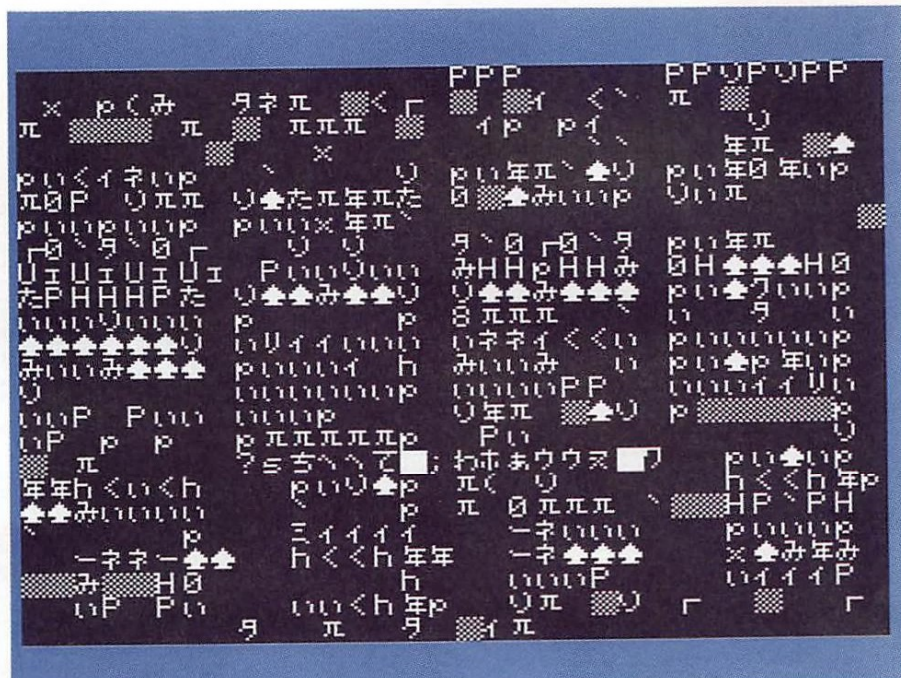
リスト3の実行後に、

VDP(2)=0:VDP(24)=64

を実行すると、すぐ下の写真のようになる。ふつうには表示することができない4分の1の部分も、スクロールさせれば画面に表示することができるのだ。

(MORO)

VPOKE文などがない
キャラクタパターン定義には、SPRITE$関数のようにパターン定義用の命令が存在しない。そのために、VPOKEを使って直接VRAMの情報を書き換えて定義するのがふつうだ。しかし1バイトずつおこなわなければならず、データもかさばってしまう。そこでページ切り換えとPRINT文で定義するテクニックなのだ。ただし、PRINT文では表示することができない文字があるので、パターンによってはまったく使い物にならない。

VDP(24)
MSX2以降の機種では、VDP(24)に、ずらすドット数を設定すればハードウエアで縦スクロールをおこなうことができる。初期状態のVDP(24)の値は0だ。

リスト3の実行画面。「ab」と「@@@@@@」の文字の形が変わっているのがわかる

リスト3を実行した後に、ページ0に切り換えて、さらにスクロールさせて見たもの

プログラムのご相談・修理・仕立て　MORO'Sショップ

●レーティング表を作るにはプログラムでレーティング表を作りたい。いったいどのようにすればよいのですか？
(宮城県・佐藤勘太郎／?歳)

レーティング表などは、要素の次元数が2つの配列変数を用意すればかんたんに作れます。

具体的には、

DIM R(12, 9)

などのようにして、DIM文で宣言して配列変数を用意しておき、それぞれに数値を代入するのです。

例えば、攻撃力とダイス値から結果を出すレーティング表なら、

R(ダイス値, 攻撃力)=結果

のようにして、あらかじめ、すべての結果を代入しておくのです。

数値を代入するときは、結果の値をDATA文にしておき、FOR〜NEXT文で一括して読み込んでしまえば、手間が省け、修正するときなども楽にできるでしょう。

●ただ今BASICの勉強中なのですが、今読んでいる本の中にはシステム変数のBASE(n)、VDP(n)についてふれられていません。この変数はどのような使い方をするのですか？
(東京都・板谷研志／17歳)

BASICにあるシステム変数のBASE(n)は、VRAMで使われている領域の先頭アドレスを参照したり変更したりするためにある変数で、nの値によって、対応する画面モードや領域が変わります。

VDP(n)もおなじように、VDPレジスタの値を参照したり設定したりするための変数で、これもnの値によって目的とするレジスタが違ってきます。

これらのシステム変数は、nの値によって役割などが違うため、ひとくちに「VDP(n)とはなにか?」のような説明となると、残念ながらこれ以上詳しくは答えられません。

☆プログラムに関する質問などを受け付けています。MSX・FAN編集部「SB講座MORO店」まで、どしどし送ってください。

Mファンにいいたい放題！

★MSXは他の機種にくらべて劣っている点も多いが、手軽にプロミングできるのが最大の魅力だ。このまま消えてしまうにはもったいない。(富山県／榊原健悟・20歳)★ここまで追いつめられていたとは……残念です。僕もMSXではもの足りずFM-TOWNSに乗り換えたロですが、いまだに(使用回数は少ないながらも)MSXをメインマシン、FM-TOWNSをサブマシンと考えており、先日ターボRをやっと探し出し購入した次第です。厳しいかもしれませんが、いつまでもエールを送りつづけるのでがんばってください。(長崎県／杉本智・19歳)